TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

UNIVERSIDAD DE ALMERIA

ESCUELA SUPERIOR DE INGENIERÍA

Implantación de un ambiente inteligente

Curso 2018/2019

Alumno/a:

Marcos Lupión Lorente

Director/es:

Pilar Martínez Ortigosa

Agradecimientos

En primer lugar, agradezco a mi directora, Pilar Martínez Ortigosa, toda la ayuda que me ha proporcionado a lo largo del proyecto. Me ha ayudado a ver las cosas de otra forma y a darme cuenta de que se puede conseguir todo lo que uno se proponga, con gran esfuerzo y dedicación. Gracias por confiar en mí y en mi trabajo.

Gracias a mi familia, por estar siempre ahí ayudándome en la medida de lo posible y dándome ánimo y consejos cuando más lo necesitaba. Vivir con ellos a lo largo de estos cuatro años ha sido fundamental para poder salir adelante en momentos difíciles.

A mis amigos de siempre: Ángela, Alberto, Rocío y Moisés. Estoy orgulloso de ser su amigo. Aunque no nos demos cuenta, los momentos que pasamos juntos son especiales. Son una parte fundamental en mi vida y espero que siempre sea así. Me han soportado y aconsejado cuando poca gente más lo hacía. En especial, gracias a mi pareja, Ana. Podría darle las gracias por mil motivos, pero me quedo con dos de ellos: me hace sentir especial y siempre está ahí, pase lo que pase.

En último lugar y no menos importante, agradezco a mis compañeros de clase, especialmente a aquellos que se han convertido en mis amigos. Estar con ellos a lo largo de estos cuatro años ha sido fundamental para llegar a este punto. Agradezco la ayuda que me han proporcionado, vamos todos a una y la verdad es que así, todo es más fácil.

Índice general

1 Introducción **1**
- 1.1 Internet de las cosas . . . 1
  - 1.1.1 SmartHomes . . . 3
- 1.2 Especificación del proyecto. Objetivos . . . 4
- 1.3 Estructura del documento . . . 4

2 Fases del proyecto **7**
- 2.1 Planificación . . . 7
- 2.2 Ejecución . . . 8

3 Herramientas **11**
- 3.1 Lenguajes de programación . . . 11
  - 3.1.1 JavaScript . . . 11
  - 3.1.2 HTML . . . 11
  - 3.1.3 CSS . . . 11
  - 3.1.4 LaTeX . . . 12
- 3.2 Comunicación . . . 12
  - 3.2.1 Correo electrónico . . . 12
  - 3.2.2 Informes . . . 12
  - 3.2.3 Reuniones . . . 13
- 3.3 Desarrollo . . . 13
  - 3.3.1 Visual Studio Code . . . 13
  - 3.3.2 Git . . . 13
  - 3.3.3 Postman . . . 14
- 3.4 Documentación . . . 14
  - 3.4.1 HomeByMe . . . 14
  - 3.4.2 TexStudio . . . 14
  - 3.4.3 GanttProject . . . 15
  - 3.4.4 Microsoft Visio . . . 15
  - 3.4.5 Google Drive Docs . . . 15
  - 3.4.6 Google Drive Sheets . . . 15
- 3.5 Infraestructura . . . 16
  - 3.5.1 Heroku . . . 16
  - 3.5.2 MongoDB Atlas . . . 16
  - 3.5.3 Pusher . . . 16
- 3.6 Stack MERN . . . 17
  - 3.6.1 MongoDB . . . 17
  - 3.6.2 ExpressJS . . . 18
  - 3.6.3 React . . . 18

3.6.4 NodeJS 18

4 Infraestructura Sistema IoT desarrollado 19

5 Sistema IoT en la niebla o neblina 23
5.1 Infraestructura y objetivos 23
5.2 Elementos del sistema 24
5.2.1 Protocolo de comunicación 24
5.2.1.1 ZigBee 24
5.2.1.2 ZWave 25
5.2.1.3 Bluetooth 26
5.2.1.4 Wi-Fi 26
5.2.1.5 Comparativa y protocolo escogido 26
5.2.2 Sensores 27
5.2.2.1 Tipos de sensores 27
5.2.2.2 Sensores a utilizar 28
5.2.2.2.1 Motion Sensor FIBARO 29
5.2.2.2.2 Door/Window Sensor 2 30
5.2.2.2.3 Arun PM3 32
5.2.2.2.4 Door/Window Sensor 34
5.2.2.3 Localización 35
5.2.3 Controlador 37
5.2.3.1 Raspberry Pi 3 B+ 39
5.2.3.2 Módulo Razberry 43
5.3 Presupuesto 46
5.4 Resultado 47

6 Sistema Web 49
6.1 Infraestructura 49
6.2 Análisis 50
6.2.1 Requisitos funcionales 51
6.2.2 Requisitos no funcionales 59
6.3 Desarrollo 60
6.3.1 Estudio de APIs 60
6.3.1.1 Estructura Z-Way 61
6.3.1.2 APIs 61
6.3.1.3 API seleccionada 62
6.3.2 Base de datos 63
6.3.3 Funcionamiento stack MERN 66
6.3.4 Script obtención nuevas notificaciones 67
6.3.5 API desarrollada en Express y NodeJS 71
6.3.6 Sistema web 74
6.3.6.1 Estructura general 74
6.3.6.2 Iniciar sesión 75
6.3.6.3 Notificaciones de sensores 77
6.3.6.4 Gestión de sensores 82

6.3.6.5 Gestión de habitaciones 83
6.3.7 Versión móvil/tablet 86
6.4 Despliegue 87

7 Resultados, conclusiones y trabajo futuro **91**
7.1 Resultados 91
7.2 Conclusiones 91
7.3 Trabajo futuro 92

Bibliografía **95**

8 Anexo I. Diagramas de Gantt **99**
8.1 Planificación 100
8.2 Ejecución 101

9 Anexo II. Sistema web. Ventanas de usuario. **103**
9.1 Inicio de sesión 104
9.2 Cambiar contraseña 105
9.3 Página principal 106
9.4 Habitaciones - Asignar-Desasignar sensores 108
9.5 Habitaciones - Añadir habitación 109
9.6 Habitaciones - Lista habitaciones 110
9.7 Sensores - Lista 111

Índice de figuras

1.1 Ámbitos IoT 2

2.1 Duración en horas de cada fase - Planificación 7
2.2 Duración en horas de cada fase - Ejecución 9

4.1 Infraestructura general sistema IoT 19
4.2 Infraestructura propia del sistema IoT implementado 20

5.1 Equema del subsistema de computación en la niebla 23
5.2 Motion Sensor Fibaro 29
5.3 Door/Window Sensor 2 30
5.4 Sensor de presión 32
5.5 Circuitos sensor Arun PM3 32
5.6 Conexión Arun PM3 - Door/Window Sensor 33
5.7 Door/Window Sensor 34
5.8 Distancia partes Door/Window Sensor 34
5.9 Conexión externa Door/Window Sensor 35
5.10 Ubicación de los sensores en la vivienda 37
5.11 Caso de ejemplo de actividades y sensores 38
5.12 Raspberry Pi 3 B+ 39
5.13 Esquema pines GPIO 40
5.14 Escritorio Raspbian 42
5.15 Módulo Razberry 44
5.16 Conexión Razberry y Raspberry Pi 3 B+ 44
5.17 Pantalla principal Z-Way 45
5.18 Adición de dispositivo a la red ZWave 46
5.19 Cambio configuración parámetros 47

6.1 Infraestructura IoT del sistema 50
6.2 Diagrama de casos de uso del sistema web 52
6.3 Estructura Z-Way y sus APIs 61
6.4 MongoDB Atlas - Consultas y modificaciones 64
6.5 MongoDB Atlas - Conexiones 64
6.6 Documento usuario 66
6.7 Inicio de sesión 75
6.8 Usuario almacenado en la base de datos 76
6.9 Cambio de contraseña 76
6.10 Lista de habitaciones 77
6.11 Últimas actualizaciones de la habitación 80

6.12 Sensores ubicados en una habitación 81
6.13 Ventana de sensores en Z-Way 81
6.14 Pop-up Z-Way con el histórico de un sensor 82
6.15 Lista de sensores que forman parte del sistema 83
6.16 Formulario añadir nueva habitación 84
6.17 Notificación: nombre de habitación existente 84
6.18 Listado de habitaciones 84
6.19 Pop-up eliminación de habitación 85
6.20 Ubicación de sensores en habitaciones 85
6.21 Combobox habitaciones almacenadas 86
6.22 Dimensiones de diferentes dispositivos y clases de Materialize 87
6.23 Panel de control en Heroku 88
6.24 Despliegue automático 89

8.1 Planificación - Diagrama de Gantt 100
8.2 Ejecución - Diagrama de Gantt 101

9.1 Inicio Sesión - PC 104
9.2 Inicio Sesión - Móvil 104
9.3 Cambiar contraseña - PC 105
9.4 Cambiar contraseña - Móvil 105
9.5 Principal - PC 106
9.6 Principal - Móvil 106
9.7 Principal - Habitación seleccionada - PC 107
9.8 Principal - Habitación seleccionada - Móvil 107
9.9 Habitaciones - Asignar-Desasignar sensores - PC 108
9.10 Habitaciones - Asignar-Desasignar sensores - Móvil 108
9.11 Habitaciones - Añadir - PC 109
9.12 Habitaciones - Añadir - Móvil 109
9.13 Habitaciones - Lista - PC 110
9.14 Habitaciones - Lista - Móvil 110
9.15 Sensores - PC 111
9.16 Sensores - Móvil 111

Índice de tablas

5.1 Comparativa de características de protocolos de comunicación 27
5.2 Especificaciones técnicas Motion Sensor Fibaro 30
5.3 Especificaciones técnicas Door/Window Sensor 2 31
5.4 Estados posibles Door/Window Sensor 2 31
5.5 Estados posibles Door/Window Sensor 2 - Configuración Modificada 31
5.6 Estados posibles Sensor de presión 33
5.7 Especificaciones técnicas Arun PM3 33
5.8 Especificaciones técnicas Door/Window Sensor 35
5.9 Características hardware de Razberry 43
5.10 Presupuesto 47

6.1 Caso de uso - Iniciar sesión. 53
6.2 Caso de uso - Ver notificaciones en tiempo real. 54
6.3 Caso de uso - Añadir habitación. 55
6.4 Caso de uso - Eliminar habitación. 56
6.5 Caso de uso - Eliminar sensor. 56
6.6 Caso de uso - Asignar-Desasignar sensor a habitación. 57
6.7 Caso de uso - Silenciar-Activar sensor 58
6.8 Caso de uso - Cerrar sesión. 59
6.9 Lista de endpoints de la API. 74

Índice de Códigos

5.1 Instalación Z-Way 45

6.1 Uso vDev API. Ejecución de comando. 62
6.2 Uso vDev API. Obtención de notificaciones a partir de una fecha. 62
6.3 Respuesta zDev API 63
6.4 Esquema de ejemplo en Mongoose 65
6.5 Esquema Notificacion 65
6.6 Esquema Habitacion 66
6.7 Esquema Sensor 66
6.8 Esquema Usuario 66
6.9 Login en la API 68
6.10 Conexión a la base de datos 68
6.11 Consulta de notificaciones a la API 69
6.12 Respuesta Notificaciones 69
6.13 Función en bucle 70
6.14 Tarea en cron 71
6.15 Endpoint GET en la API 71
6.16 Objeto JSON con respuesta de la API 72
6.17 Endpoint POST en la API 72
6.18 JSON de datos para la consulta POST 72
6.19 Endpoint DELETE en la API 73
6.20 Comprobación de contraseña 76
6.21 Comprobación logueado 77
6.22 Creación de componentes hijos 78
6.23 Creación objeto Pusher 78
6.24 Restricciones función watch() 79
6.25 Inserción de notificación en el canal 79
6.26 Llamada a la API 80
6.27 Aviso finalización streaming 80
6.28 Detención de la ejecución de watch() 80
6.29 Componente Pagination 82
6.30 JSON de la llamada POST 83

1 Introducción

1.1 Internet de las cosas

Desde los inicios de Internet, el objetivo de este conjunto de redes interconectadas era ofrecer diferentes servicios a las personas que faciliten el día a día de estas. Con el nacimiento de esta tecnología, se desarrollaron muchísimos avances (mensajería instantánea, aplicaciones de gestión empresarial, televisión online, navegadores web, etc.), favoreciendo la conexión de cada vez más dispositivos a esta gran red para hacer uso de estos servicio.

Hasta hace no mucho tiempo, aproximadamente hacia el año 2014, solamente se conectaban a Internet dispositivos como ordenadores, smartphones o tablets, alcanzando un total de 12 mil millones de dispositivos, el equivalente a 1,7 dispositivos por persona[1]. Sin embargo, a partir de ese año, la cantidad de dispositivos conectados a Internet comenzó a incrementar de forma exponencial ya que los dispositivos que se empezaron a conectar eran diferentes a los que se conectaban previamente.

Estos dispositivos se tratan de objetos cotidianos como relojes, luces, lámparas o puertas que se conectan a Internet y proporcionan información sobre el estado en el que se encuentran. Esta nueva información que aportan puede ser usada para ofrecer diferentes servicios a los usuarios. Así, hacia el año 2020, se estima que se conectarán a Internet hasta 50 mil millones de dispositivos, siendo la mayoría de estos, objetos cotidianos usados en el día a día.

Este enfoque de conectar dispositivos a Internet no es algo totalmente nuevo. Por ejemplo, en el año 1982, en la Universidad de Carnegie Mellon se desarrolló una máquina de venta de refrescos[2] que incorporaba conexión a Internet. De esta forma, los usuario podían comprobar cuándo había nuevos refrescos fríos en esta y proceder a comprarlos. Este enfoque es el claro ejemplo de lo que se conoce como Internet de las Cosas.

Así, Internet de las Cosas consiste en la conexión de objetos cotidianos a Internet con el objetivo de proporcionar y recibir datos de Internet para facilitar la vida al usuario. Internet de las Cosas a su vez tiene diferentes ámbitos de aplicación, dependiendo del tipo de dispositivos que se conecten a Internet y de los servicios que se ofrecen. Así, los principales campos (daCosta y Henderson, 2013) en los que se aplica esta tecnología se muestran en la Figura 1.1.

En primer lugar, IoT ha sido rápidamente expandido en el dominio industrial, incorporando todo tipo de dispositivos para agilizar y tener un mayor control sobre la producción, incremen-

[1] https://www.muycomputer.com/2014/10/24/mas-de-12-000-millones-de-dispositivos-estaran-conectados-internet-en-2015/

[2] https://www.cs.cmu.edu/ coke/

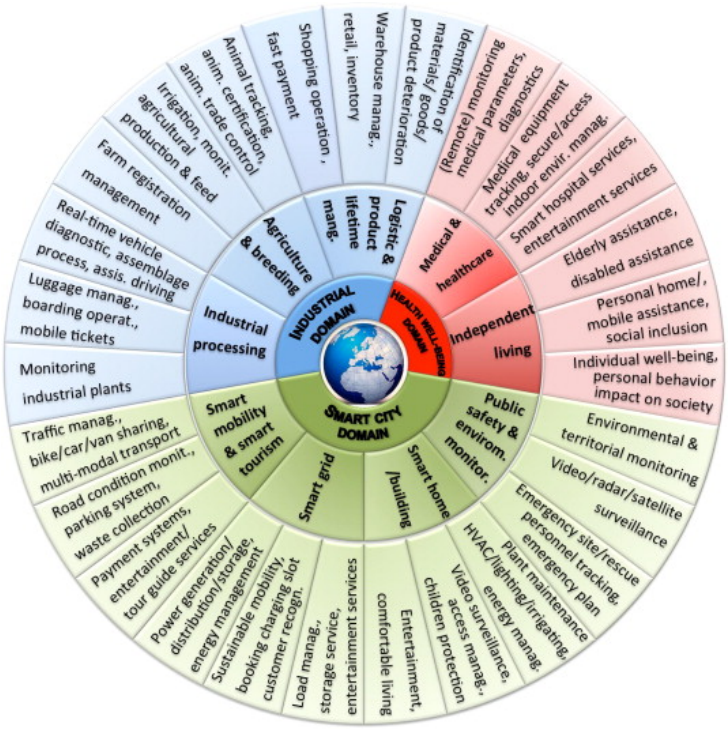


Figura 1.1: Ámbitos IoT

tando la facturación y beneficios de las industrias o empresas que han aplicado esta tecnología. La aparición de IoT en este ámbito ha significado una pequeña *revolución industrial*, cambiando los procesos productivos mediante la incorporación de esta nueva tecnología.

En segundo lugar, otro ámbito que destaca se trata de las ciudades inteligentes. En este campo, se dota a los dispositivos de las ciudades y casa de conexión a internet para informar sobre el estado de estos dispositivos o posibles fallos que ocurran, favoreciendo la seguridad y la calidad de vida principalmente. De esta forma, en un futuro, el número de ciudades inteligentes crecerá de forma exponencial[3], ya que el número de ciudades cada vez es mayor y el número de habitantes en estas también, por lo que la incorporación de IoT en este ámbito favorecerá la vida de las personas que viven en estas. Estas ciudades inteligentes proporcionan diferentes servicios adaptados a los ciudadanos, como por ejemplo la visualización del estado del tráfico, alertas sobre fenómenos temporales, indicadores de contaminación, geolocalización de transporte público, etc.

En último lugar, otro ámbito en el cual se ha avanzado en el uso de IoT se trata del ámbito sanitario. Gracias a esta tecnología, se permite controlar la salud de las personas mediante pequeños sensores y aplicaciones que miden constantemente ciertas magnitudes físicas y previenen algunas enfermedades o ataques.

El trabajo a realizar se centra en la construcción de un sistema IoT relacionado con el ámbito

[3]https://hablemosdeempresas.com/grandes-empresas/futuro-de-las-smart-cities/

de las *smart cities*. Concretamente, se construye un sistema IoT aplicado a una vivienda con el objetivo de monitorizar los eventos que se producen en ella. En la siguiente sección se profundiza sobre este ámbito.

1.1.1 SmartHomes

Dentro del campo de aplicación de Internet de las Cosas, se encuentran las *smart homes*. Este concepto hace referencia a *viviendas inteligentes*, entendiendo por *inteligente* el hecho de estar compuestas por una gran variedad de sensores y actuadores capaces de realizar ciertas acciones de forma automatizada sin la intervención del usuario con el objetivo de mejorar la experiencia del usuario en la vivienda (Ding y cols., 2016).

El concepto de *smart home* puede tener su origen hacia la mitad de la década de los años 90[4], cuando Bill Gates, fundador de Microsoft, incorporó a su casa diferentes dispositivos electrónicos con el objetivo de que la vivienda aprendiese de los hábitos y gustos del usuario para adaptar la vivienda a este (Gentry, 2009).

En la actualidad, las *smart homes* no solamente tienen como objetivo adaptar las características de la vivienda al usuario o realizar tareas de ahorro energético (las habitaciones vacías no se quedan con la luz encendida, se suben las persianas para no encender la luz, etc.), sino se están usando como medio de control y monitorización de personas mayores o con enfermedades. El objetivo de las *smart homes* en estos casos es proporcionar las herramientas y facilidades a estos usuarios para que puedan llevar a cabo una vida independiente en sus propias casas, sin tener que recurrir a la contratación de personal externo.

El uso de *smart homes* enfocadas en personas mayores está incrementando en la actualidad (Hoof y cols., 2016), ya que se ha producido un aumento del 131 por ciento de personas mayores de 64 años en España desde hace una década[5]. Esto hace que cada vez hay más gente mayor en España y las *smart homes* adaptadas a esta población están en constante evolución (Chana y cols., 2009). En este campo, las *smart homes* incorporan una gran cantidad de sensores situados en la vivienda y en el usuario, como acelerómetros para detectar caídas, GPS para controlar la ubicación del usuario, sensores de presencia, etc. para logar una completa monitorización de estos usuarios y reaccionar lo más rápido posible ante algunas circustancias como caídas (Shumei y cols., 2017) (Yin y cols., 2016)

Actualmente, se están realizando investigaciones sobre las necesidades de un usuario en una vivienda inteligente, con el objetivo de adaptar las características de las viviendas a estos.

[4] https://www.digitaltrends.com/home/the-awesome-technology-inside-bill-gates-mansion/
[5] https://www.noticiasdegipuzkoa.eus/2019/01/29/sociedad/estado/el-numero-de-mayores-de-64-anos-se-incrementa-en-espana-un-131-en-una-decada

1.2 Especificación del proyecto. Objetivos

El proyecto a desarrollar consiste en la implementación de una *smart home* usando sensores distribuídos por la vivienda (Yin y cols., 2016). Estos, recogen datos del estado en el que se encuentra la vivienda con el objetivo de ser analizados por el usuario y este pueda llevar a cabo acciones de ahorro energético, la implementación de sistemas de seguridad, etc. Para implementar el sistema se hace uso de sensores no intrusivos para el usuario, con el objetivo de preservar la intimidad de este (Ding y cols., 2011). Además, se desarrolla una herramienta web que permite al usuario la obtención de estos datos en los dispositivos móviles y desde fuera de casa, permitiendole a este una monitorización del estado de la vivienda en todo momento.

Los objetivos que se persiguen con la realización del proyecto son los que se muestran a continuación:

- Configuración e instalación de sensores en la vivienda con el objetivo de abarcar el máximo número de superficie posible.
- Visualización de la información que proporcionan estos sensores instalados haciendo uso de un dispositivo central que coordine toda la infraestructura implementada.
- Monitorización continua de los cambios que se producen en los sensores desde dentro y fuera de la vivienda mediante la realización de un sistema web propio que accede a los datos generados por los sensores.
- Adaptación del software a las necesidades del usuario partiendo de los datos de los sensores que se disponen.

Estos objetivos por tanto buscan la creación de una vivienda inteligente adaptada al usuario final.

1.3 Estructura del documento

En este capítulo se ha realizado una introducción al tema del trabajo realizado, así como la motivación del estudiante para realizar un proyecto de este tipo. Además, se especifica el proyecto desarrollado junto con los objetivos que se persiguen al realizar este.

En cuanto al capítulo 2, se realiza una descripción de las fases en las cuales se ha dividido el proyecto. Además, se muestran varios diagramas en los que se muestra la diferencia entre la planificación del proyecto y la ejecución de este.

En el capítulo 3, se describen las herramientas que se han usado para realizar el proyecto. Dentro de estas herramientas se engloban los lenguajes de programación, las herramientas de comunicación, los entornos de desarrollo y herramientas de apoyo al desarrollo de software, las herramientas que han servido para realizar la documentación del proyecto, los servicios que se han usado para realiza la infraestructura web del proyecto y, por último, las tecnologías que fundamentan el stack MERN, fundamental para el desarrollo del sistema web.

A continuación, en el capítulo 4 se expone la teoría de los sistemas IoT de forma general y se especifica el sistema IoT completo desarrollado en el proyecto.

En el siguiente capítulo, el capítulo 5, se muestra una parte del sistema IoT implementado, concretamente, el subsistema de IoT en la niebla o neblina. Aquí se muestran los resultados del estudio y configuración de los sensores y software que forma este subsistema.

Posteriormente en el capítulo 6, se muestra la infraestructura del subsistema IoT que engloba el sistema web, junto con la página web desarrollada para realizar la monitorización de los sensores.

En el último capítulo, el capítulo 9, se exponen las conclusiones obtenidas al desarrollar el sistema, así como el trabajo futuro en este proyecto.

Finalmente, se muestra la bibliografía de la que se ha hecho uso a lo largo de la redacción de la documentación y varios anexos.

2 Fases del proyecto

Para conseguir la correcta consecución de los objetivos que se han planteado anteriormente, se dividió el proyecto en diferentes fases. Así, en este apartado se incluyen las fases en las cuales se ha dividido la realización del proyecto junto con la cantidad en horas que ha durado cada una y un diagrama de Gantt en el que se muestra su distribución a lo largo del tiempo.

2.1 Planificación

En un primer lugar, se realizó una planificación de las tareas a realizar en el proyecto junto con su duración estimada en horas (Figura 2.1). En total, se estimó una duración de 274 horas para el proyecto a realizar.

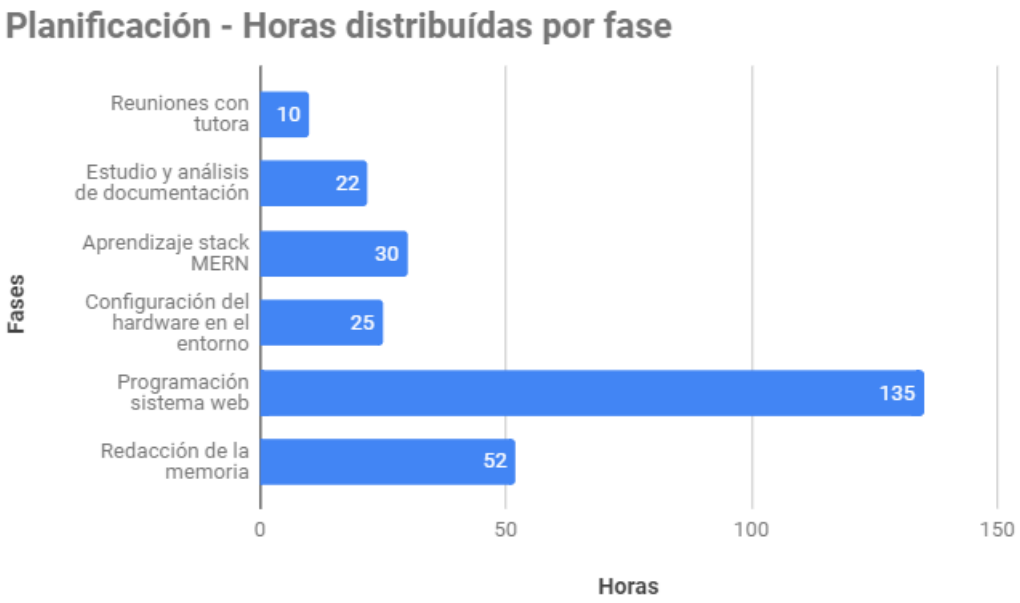


Figura 2.1: Duración en horas de cada fase - Planificación

Una vez se establecieron estas, se distribuyeron a lo largo del tiempo con el objetivo de distribuirlas de la forma más eficiente, como se puede observar en el diagrama de Gantt del Anexo 8.1.

La estimación realizada sirvió para tener una visión global de las tareas a realizar y distribuirlas en el tiempo para ejecutar el proyecto correctamente. Sin embargo, dado que las tareas que se

definieron eran de gran tamaño, posteriormente se dividieron en subtareas, como se ha podido observar en el diagrama de Gantt.

2.2 Ejecución

En esta sección se muestran las fases en las que se dividió el proyecto para su correcta realización. Así, se realiza una descripción de las tareas que se han llevado a cabo en cada una de ellas junto con su duración en horas y su distribución a lo largo del tiempo.

Reuniones con la tutora (10 horas) En esta primera fase, se sucedieron diferentes reuniones con la tutora a fin de establecer los requisitos y la temática del proyecto. Además, una vez que queda claro el tema del proyecto, se obtuvo el hardware necesario para comenzar con el proyecto.

Estudio sobre el campo de aplicación del proyecto (19 horas) Para comprender el campo de IoT, se estudió diferente bibliografía sobre este tema. Además, se realizaron varios cursos CISCO con el objetivo de profundizar más aún en los componentes y objetivos de la tecnología IoT.

Estudio y análisis de documentación asociada al hardware obtenido (24 horas) Una vez que se obtiene el hardware, se estudió y comprendió cómo se interactúa con este hardware. Para ello, se estudiaron las diferentes APIs y software que se proporciona para establecer esta interacción con el hardware.

Implementación de sistema en la niebla o neblina (37 horas) Una vez que se conocía el funcionamiento del hardware obtenido, se procedió a ubicar los sensores en la vivienda. Para la ubicación de estos, se tuvo que realizar un estudio sobre la distribución de los mismos con el objetivo de abarcar la máxima superficie posible con estos sensores y obtener la información más relevante de la vivienda. Además, se tuvieron que configurar los distintos dispositivos que forman parte de este sistema.

Uso de las APIs (15 horas) Después de la implementación del sistema en la niebla o neblina, se comprobó que las APIs que proporciona el hardware funcionan correctamente. Así, se decidió usar una de ellas para obtener los datos de los sensores de forma sencilla. Una vez que se realizó la conexión con estas de forma satisfactoria y se extrayeron los datos necesarios para realizar el sistema web, se comenzó a realizar la siguiente tarea.

Realización del sistema web (132 horas) Cuando se comprendió cómo interactuar con el hardware proporcionado, se buscaron las tecnologías software para llevar a cabo esta comunicación con el hardware. El sistema que se usa para interactuar con el software se trata del stack MERN (MongoDB, Express, React y NodeJS). Una vez que se decide usar este sistema a la vista de sus ventajas, se tuvo que aprender a usarlo para poder adaptarlo al problema que se tiene. Así, fue necesaria la visualización de videotutoriales y estudio de la documentación de estas tecnologías para poder desarrollar un sistema web completo. Cuando se aprendió a usar

la pila de tecnologías mencionadas, se procedió a desarrollar el sistema web. En primer lugar se realizó la interfaz para PC y posteriormente se adaptó a dispositivos móviles.

Redacción de la memoria (53 horas) A lo largo de la realización del proyecto, se fueron redactando algunas partes de la memoria, concretamente algunos aspectos fundamentales que se descubrían a medida que se realizaba el proyecto. Así, una vez finalizada la implementación del proyecto, se recopiló toda esta información y se plasmó en el documento con una visión más global.

En la Figura 2.2 se puede observar las horas que se han empleado para cada fase. Como se puede apreciar en esta, la fase en la cual se ha desarrollado el sistema web ha sido la fase que más tiempo ha llevado, con 132 horas. En total, la duración del proyecto es 290 horas.

Figura 2.2: Duración en horas de cada fase - Ejecución

La distribución de las tareas en el tiempo se puede observar en el diagrama de Gantt del Anexo 8.2.

3 Herramientas

En este capítulo se va a realizar una descripción de las herramientas que se han usado para desarrollar el trabajo. Además de la descripción de estas herramientas, se va a explicar por qué se ha decidido usar cada una de ellas en lugar de cualquier otra. Para organizar la descripción de las herramientas, estas se han dividido en cuatro categorías.

3.1 Lenguajes de programación

En esta categoría se describen los lenguajes de programación que se han usado para realizar el proyecto.

3.1.1 JavaScript

JavaScript se trata de un lenguaje de programación orientado al desarrollo web. Este lenguaje se basa en *C*, y se trata de un lenguaje orientado a objetos. Al principio, se usaba solamente en el lado del cliente para dinamizar las vistas que se muestran a este; sin embargo, también permite actuar en el lado del servidor, como se puede apreciar en el *framework* NodeJS.

El sistema web que se ha desarrollado está basado en NodeJS, ReactJS y Express. Estos *frameworks* están basados en JavaScript, por lo que ha sido el lenguaje predominante en el desarrollo del sistema web. Se ha decidido hacer uso de este lenguaje ya que se trata de un lenguaje optimizado para el desarrollo de sistemas web, el cual puede ser utilizado para desarrollo *back-end* y *front-end*.

3.1.2 HTML

Se trata de un lenguaje de programación basado en etiquetas cuyo principal objetivo es la creación de páginas web. Se trata de un estándar dentro del desarrollo web, ya que fue establecido por *W3C*[1]. Todos los navegadores soportan este lenguaje, ya que se trata del lenguaje que se usan para mostrar las ventanas al usuario.

Dado que se ha desarrollado un sistema web, se ha usado este lenguaje de programación para realizar el *front-end* del mismo. Se ha decidido usar este lenguaje porque no hay otra opción de cara a realizar el la estructura de etiquetas de un sistema web.

3.1.3 CSS

Se trata del lenguaje de programación usado para personalizar y diseñar el estilo de las ventanas que se incluyen en el sistema web. Este lenguaje surgió con la necesidad de separar el

[1] World-Wide-Web Consorcium

contenido de una página web o documento de su visualización, por lo que con este lenguaje se permite personalizar la visualización tanto de documentos como de páginas web. Actualmente, se trata del lenguaje más usado para establecer los estilos de las páginas web.

Dada la gran cantidad de ventanas a desarrollar, ha sido necesario hacer uso del *framework* MaterializeCSS. Este *framework* incluye un conjunto de clases CSS que se pueden aplicar a los elementos incluidos en formato HTML. Todas estas clases están realizadas siguiendo el diseño *Material Design* de Google tan usado en la actualidad. Además, este *framework* incluye funciones descritas en JavaScript que favorecen el dinamismo de estos elementos.

3.1.4 LaTeX

LaTeX se trata de un lenguaje de tipografía de alta calidad. Con este lenguaje se pueden crear documentos con bastante grado de personalización y calidad, como por ejemplo documentos científicos y publicaciones. Este lenguaje está formado por órdenes construidas a partir de comandos *TeX*, por lo que se trata de un lenguaje de bajo nivel. Además, se trata de un software de código abierto, por lo que está en constante evolución y mejora.

Dada su gran aceptación en la redacción de trabajos de investigación y memorias de proyectos, se ha hecho uso de una plantilla para redactar la memoria final del proyecto. Además, se ha decidido usar este lenguaje por la facilidad con la que se pueden personalizar los elementos que se incluyen en el documento.

3.2 Comunicación

En esta categoría se engloban los medios a través de los cuales se ha realizado la comunicación entre la tutora y el alumno a lo largo del desarrollo del proyecto.

3.2.1 Correo electrónico

Se ha hecho uso de esta herramienta para el intercambio de mensajes entre alumno y tutora. Gracias a esta herramienta, se tiene un historial de los mensajes intercambiados, pudiendo acceder a la diferente información que se ha tratado en ellos a lo largo del tiempo y siendo fácilmente accesible por ambas partes. También se ha usado esta herramienta para fijar reuniones y tutorías en lugar de usar alguna otra plataforma de mensajería instantánea.

3.2.2 Informes

A lo largo de la realización del trabajo, se han realizado varios informes para exponer ideas sobre el sistema a desarrollar, así como para el resumen y envío de bibliografía. Gracias a la realización de estos informes, se ha tenido una gran cantidad de información documentada lista para incluir en la memoria final.

3.2.3 Reuniones

Han sido necesarias algunas reuniones con la tutora para definir el ámbito del proyecto y resolver dudas que han ido surgiendo a lo largo del mismo. Se trata de la forma más efectiva de comunicación posible entre ambas partes, ya que la interacción se realiza en persona.

3.3 Desarrollo

En esta categoría se realiza una pequeña descripción de los entornos de desarrollo y herramientas usados para realizar la programación del trabajo. Además, se justificará su uso en lugar de otro software con funcionalidad similar.

3.3.1 Visual Studio Code

Se trata de un *IDE* de código abierto, enfocado a la programación en JavaScript dada su rapidez y capacidad de personalización. La principal característica de este software consiste en instalar y configurar complementos para personalizar la experiencia del usuario cuando se programa. A pesar de estar enfocado para el desarrollo de código web (en JavaScript), se puede usar para cualquier lenguaje de programación, pudiendo instalar complementos de todos los lenguajes y adaptándose a estos.

Se ha decidido hacer uso de Visual Studio Code para la codificación del sistema web completo. Dado que este está basado en JavaScript, se ha usado Visual Studio Code ya que se trata de un *IDE* optimizado para este lenguaje de programación. Este incluye numerosos *plugins* que permiten personalizar al usuario los diferentes aspectos del código como su visualización y formato, lo cual ha facilitado la programación. Una funcionalidad importante ha sido poder crear áreas de trabajo, permitiendo tener varios proyectos abiertos a la vez, favoreciendo el intercambio de archivos entre ellos.

3.3.2 Git

Git se trata de un sistema de control de versiones de código abierto. Además, se trata de un sistema de control de versiones distribuído, ya que no solamente se tiene un repositorio central en el cual van a estar todos los cambios, sino que se tiene un repositorio central y otro repositorio para cada usuario que participe en este; de esta forma, se tiene una copia de los datos para cada usuario y solamente se compartirán los datos que cada usuario quiera, siendo esta subida/bajada de datos una operación bastante rápida ya que no se tiene que hacer sobre todo el contenido, sino sobre estas modificaciones que se hagan.

Se ha hecho uso de esta herramienta para realizar un control de código fuente sobre el sistema que se ha desarrollado. Concretamente, se ha usado *GitHub*, que permite tener repositorios privados de forma gratuita y actualmente se trata de la herramienta más usada para control de versiones Git. Gracias a este sistema se ha podido volver a versiones anteriores del código cuando ha sido necesario. Además, se ha usado ya que los *IDEs* que se han comentado anteriormente tienen una integración directa con este software, por lo que se han podido subir, actualizar y revertir los cambios de forma sencilla.

3.3.3 Postman

Se trata de una herramienta desarrollada para realizar peticiones *HTTP* a cualquier *API*[2]. De esta forma, se puede testear fácilmente una *API* que se esté desarrollando o cualquier *API* disponible en la red. Esta herramienta está bastante optimizada y en este momento, numerosos *IDEs* están intentando incorporarla de forma nativa.

Aunque algún *IDE* incluye esta herramienta de forma nativa, se ha hecho uso de este *software* porque permite la creación de peticiones a APIs internas y de terceros, además de la creación de entornos de trabajo con variables globales y locales. Además, gracias a la facilidad de uso de esta herramienta, se ha ahorrado tiempo testeando la *API* que se ha desarrollado, al no ser necesaria la realización de consultas a esta desde el sistema desarrollado, sino directamente desde Postman.

3.4 Documentación

En esta sección se muestran las herramientas que se han usado para realizar la documentación del trabajo realizado.

3.4.1 HomeByMe

Se trata de una aplicación web de diseño de interiores que permite la creación de diseño de interior de viviendas en *3D*. Además, incluye una gran cantidad de elementos del mobiliario de una vivienda, por lo que se pueden crear diseños realistas que incluyan estos elementos.

Se ha usado esta aplicación web principalmente porque se trata de un recurso en línea, y no hace falta descargar ninguna aplicación de escritorio para poder usarlo. Además, se trata de una aplicación específica para diseño de interiores, que es lo que se trata de hacer en este caso, no ha sido necesario adaptar una aplicación orientada al diseño en general (Autocad) para hacer los diseños del interior de la vivienda que se quieren hacer.

3.4.2 TexStudio

Se trata de un editor de código LaTeX , con todas las ventajas que aportan los editores de textos. Además, permite una visualización de los cambios del documento en tiempo real, y la corrección y muestra de errores de sintaxis a medida que se va generando el documento.

Esta herramienta se ha usado porque se permite su uso en Windows, sistema operativo en el cual se ha trabajado a lo largo de la realización del proyecto. Además, está especialmente diseñado para la creación de documentos en lenguaje LaTeX , usado para la redacción de la memoria final.

[2]Interfaz de programación de aplicaciones

3.4.3 GanttProject

El diagrama de Gantt se trata de una de las herramientas más usadas para la gestión de proyectos. Con este diagrama se realiza la planificación de las tareas a lo largo del tiempo con el objetivo de realizar el proyecto de forma correcta. Esta herramienta fue inventada por Henry L. Gantt entre los años 1910 y 1915. Gracias a esta herramienta, se puede ver de una manera general el tiempo que se ha invertido en cada una de las tareas así como el orden en el que se han realizado y las precedencias de estas entre sí.

El software Gantt Project se trata de un software de código abierto disponible para Windows, OSX y Linux que permite la realización de estimaciones en forma de diagramas de Gantt. Se ha decidido hacer uso de esta herramienta por su facilidad de uso y su gratuidad. Además, se ha encontrado de bastante utilidad la opción de comparar la estimación con la ejecución del proyecto, permitiendo ver las diferencias entre ambas.

3.4.4 Microsoft Visio

Microsoft Visio se trata de una herramienta que permite la realización de una gran cantidad de diagramas.

Se ha hecho uso de esta herramienta ya que se trata de un software proporcionado por Microsoft de forma gratuita para estudiantes y permite realizar específicamente diagramas de casos de uso, con todos los elementos necesarios para ello.

3.4.5 Google Drive Docs

Actualmente, Google Drive, servicio de Google en la nube para el almacenamiento y gestión de archivos creado en 2012, permite la creación de archivos en la nube con un editor similar al proporcionado por herramientas como Word. La heramienta con la que se puede realizar esto es Google Drive Docs. En esta aplicación en la nube se permite la creación, modificacion, compartición y almacenamiento de documentos , con todas las ventajas que ellos conlleva. Entre algunas de las ventajas se encuentran por ejemplo, el trabajo en grupo sobre un documento o el guardado automático con cada cambio que se realice en el archivo.

Se ha hecho uso de esta herramienta ya que se trata de una utilidad que se encuentra disponible de forma gratuita en la nube. Así, no requiere instalación y se mantienen los documentos actualizados en todo momento, no siendo necesaria la copia de seguridad de estos documentos. Además, permite tener un historial de los cambios y volver atrás cuando se requiera. Se ha usado para la redacción preliminar de la memoria final y la redacción de distintas anotaciones a medida que se iba realizando el proyecto.

3.4.6 Google Drive Sheets

Esta herramienta está integrada también en Google Drive. Así, permite la creación de hojas de cálculo tal y como hace Excel, pero con todas las ventajas de estar en la nube y en el mismo paquete que Google Drive Docs. De esta forma, los archivos que se generan en las herramientas

de Google Drive tienen un alto grado de acoplamiento entre ellos.

Esta utilidad está incluída dentro del mismo paquete que la herramienta anterior, con todas las ventajas mencionadas anteriormente. En este caso, se ha usado esta herramienta para la realización del historial de actividades y tareas, junto con la duración de las mismas, que se han realizado para el completar el proyecto.

3.5 Infraestructura

A continuación se va a realizar una descripción de los servicios que se han usado para configurar la estructura IoT.

3.5.1 Heroku

Se trata de un servicio PaaS[3], que permite el despliegue de aplicaciones de forma sencilla, y casi automática, liberando al administrador de aplicaciones de las tareas de configuración y entorno de las diferentes aplicaciones. De esta forma, se permite un rápido despliegue de aplicaciones indicando únicamente la tecnología de back-end que se va a usar.

Ha sido necesario usar Heroku para desplegar el sistema web que se ha realizado para que pueda ser accesible desde cualquier lugar, no solamente desde la universidad o desde la red local de casa. Así, se ha accedido desde fuera de casa al sistema para poder realizar el entrenamiento del algoritmo mediante la corrección e introducción de datos derivados de acciones realizadas en la vivienda. Además, se trata de una herramienta gratuita siempre que no se superen los 500mb de almacenamiento.

3.5.2 MongoDB Atlas

Se trata de un servicio DaS[4], que permite la creación de *clusters* de bases de datos en la nube (con todas las ventajas que esto conlleva), pudiendo acceder a ellas desde cualquier aplicación que se desarrolle. Este servicio se encarga de la configuración e instalación de las base de datos en los clusters, por lo que las tareas del administrador solamente consisten en usar estas bases de datos.

Se ha usado este servicio de base de datos en la nube debido a que ha sido necesario crear una base de datos común a varias aplicaciones y *scripts*, por lo que la única forma de conseguir esto ha sido haciendo uso de este servicio, que proporciona bases de datos MongoDB (el tipo de bases de datos usadas en el proyecto). Además, dado que esta plataforma tiene varios planes de pago, se ha usado el plan gratuito, que se caracteriza principalmente por tener una cantidad de memoria de 512 MB y la memoria RAM compartida con otros clusters.

3.5.3 Pusher

Pusher se trata de un servicio en la nube que proporciona funcionalidades de tiempo real a aplicaciones. Este servicio encapsula la aplicación del usuario, no siendo necesario ejecutar un

[3]Plataforma como servicio
[4]Base de datos como servicio

servidor de Websockets para realizar streaming de datos. Con la versión gratuita se permiten hasta 200.000 mensajes enviados al día y un número máximo de 100 conexiones a la vez.

Se ha hecho uso de este servicio por su facilidad de uso y extensa documentación. Además, la versión gratuita es más que suficiente para abarcar las necesidades del proyecto desarrollado.

3.6 Stack MERN

El stack MERN consiste en un conjunto de tecnologías relacionadas entre sí que tienen como objetivo favorecer la creación de soluciones web de forma eficiente y sencilla para el usuario. Así, dado que MERN se trata de un stack, estas herramientas tienen funcionalidad independiente entre sí, pero se relacionan para proporcionar una solución global, en este caso, una solución para el desarrollo completo de sistemas web, englobando tanto el front-end como el back-end. Un stack que engloba tanto el front-end como el back-end se trata de un full stack.

El uso de stack para el desarrollo web no es nuevo, existen stacks como LAMP (Linux, Apache, MySQL, y PHP) o MEAN (MongoDB, Express, AngularJS y Node.js) anteriores a MERN (MongoDB, Express, React y Node.js), por lo que MERN trata de reunir las ventajas de estos stacks antiguos para formar un stack más moderno y potente.

Así, MERN apareció por la rápida popularización de React gracias a Facebook. React se trata de la librería de front-end que usa Facebook por lo que se intercambió el framework de front-end AngularJS por React para formar el stack MERN.

Sin embargo, la característica de MERN es que todas las tecnologías que forman parte de este, usan el lenguaje de programación JavaScript. De esta forma, no se tiene que programar el front-end con una tecnología y el back-end con otra diferente, como en la mayoría de los stack de programación web.

3.6.1 MongoDB

MongoDB se trata de la base de datos usada por MERN. Esta base de datos Open Source fue creada en 2007 por la compañía MondoBD Inc. Así, MongoDB se trata de una base de datos noSQL, es decir, no está formada por tablas y columnas, sino por colecciones y documentos. Así, estos documentos están formados en el lenguaje JSON, por lo que permite un fácil manejo de estos. Una característica fundamental de esta base de datos es que no tiene un esquema predefinido, puede haber en una misma colección documentos con diferentes campos, permitiendo una gran flexibilidad en la información que almacena.

Se ha querido hacer uso de esta base de datos ya que en los sistemas IoT se tiene mucha información que almacenar, siendo necesario su procesado y consulta, por lo que esta base de datos al no estar estructurada y permitir la indexación, realiza las consultas con mayor velocidad que las bases de datos relacionales.

3.6.2 ExpressJS

Se trata de un framework que permite simplificar las tareas de escribir código para el servidor que maneja el sistema web. Dado que el servidor contiene código Javascript ejecutado por NodeJS, Express proporciona las herramientas necesarias para garantizar el correcto funcionamiento del servidor de NodeJS. Así, Express permite la creación de rutas para definir una API HTTP, manejando toda la información que se incluyen en estas consultas HTTP, como puede ser los headers, cookies y respuestas del servidor. En definitiva, Express proporciona un framework web para usar NodeJS como servidor de una aplicación web.

3.6.3 React

React se trata de una librería de JavaScript Open Source creada por Facebook en el año 2013. Esta librería se caracteriza por permitir la creación de componentes, elementos que gestionan su estado y tienen funcionalidad propia, pudiendo interactuar con otros componentes padres o hijos con los que estén relacionados. Además, React maneja vistas declarativas, por lo que el usuario no tiene que preocuparse de los efectos que se producen cuando una vista cambia de estado debido a cambios en los datos de esta. En definitiva, incluye funciones que normalmente están incluídas en jQuery, pero de forma nativa.

Se ha usado React con el objetivo de lograr un alto grado de separación de funcionalidad entre las vistas, además de proporcionar código de jQuery de forma nativa, sin tener que preocuparse por la visualización de los elementos y el contenido dinámico del sitio web.

3.6.4 NodeJS

NodeJS se trata del entorno que permite la ejecución de código JavaScript independientemente del navegador. Así, se pueden ejecutar *scripts* de JavaScript en el terminal local de un PC usando NodeJS. Este entorno permite la instalación de librerías para incluir funcionalidad a los *scripts* de JavaScript que se ejecuten. De esta forma, se pueden realizar tareas de programación que tradicionalmente han sido realizadas en lenguajes como Python o C, usando JavaScript. Actualmente está muy de moda para el desarrollo de servidores web, permitiendo realizar las funciones de back-end en JavaScript, lenguaje que nunca había traspasado las fronteras del front-end.

Se ha decidido usar NodeJS para la ejecución de diferentes *scripts* en el controlador de IoT que se ha incluído en el proyecto, además de para el desarrollo del sistema web.

4 Infraestructura Sistema IoT desarrollado

El sistema que se va ha creado se trata de un sistema basado en internet de las cosas (IoT). Este concepto puede dar lugar a confusión, ya que a veces se implementan sistemas IoT que no hacen uso de internet, sino que se tratan de un conjunto de sensores conectados a un controlador, el cual recibe los datos, los organiza y ejecuta ciertas acciones dependiendo del caso. Para hacer esto, no es necesario usar internet. Esto se llama internet de las cosas en modo computación en la niebla. La diferencia entre este modo de computación y la computación en la nube, es que en el modo de computación en la nube, el controlador tiene acceso a internet y desde internet se realizan acciones como almacenamiento, procesamiento o simplemente visualización de los datos procedentes del controlador.

Así, la infraestructura general de los sistemas de internet de las cosas (Figura 4.1) se basa en las capas que se justifican en (Cisco, 2018-2019). Este enfoque se basa en una infraestructura general. En (Hanes y cols., 2017), en lugar de las 4 capas que se especifican a continuación, se divide en 7 capas, diferenciando especialmente la capa de la nube en subcapas con diferente funcionalidad en IoT.

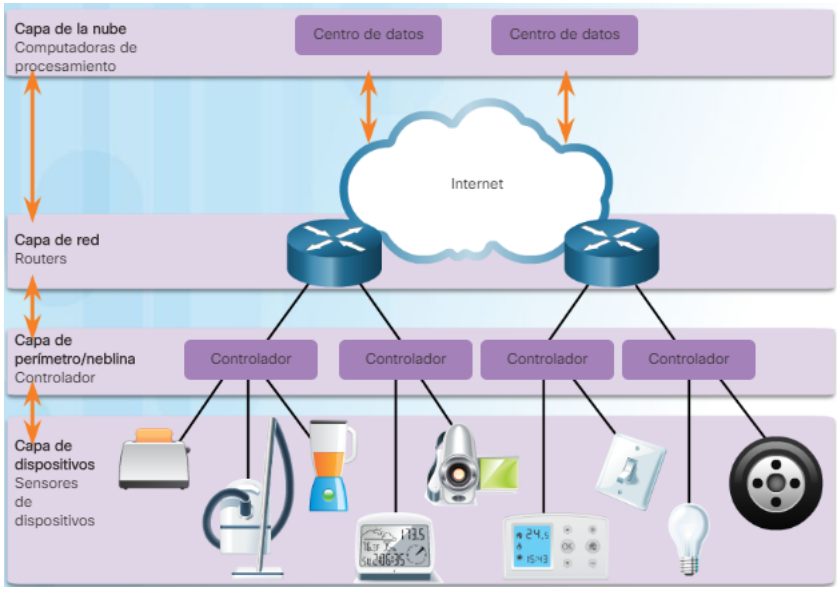


Figura 4.1: Infraestructura general sistema IoT

Capa de dispositivos físicos: Esta capa está compuesta por un conjunto de dispositivos que proporcionan información sobre el entorno.

Capa de neblina o niebla: En esta capa se encuentra el dispositivo o dispositivos que se encar-

gan de obtener los datos que son proporcionados por los sensores de la capa de dispositivos físicos.

Capa de red: En esta capa se realiza la conexión entre el controlador, dispositivo que contiene la información recibida por los sensores, e internet.

Capa de la nube: En esta capa se encuentran las tecnologías que procesan o almacenan los datos generados por los sensores u otra información.

Dado que este trabajo nace con el objetivo principal de crear una estructura IoT en la nube, la computación y procesamiento principal de los datos de los sensores no se realiza en el dispositivo que ejerce de controlador, sino que estos datos son procesados en la nube para generar conocimiento como el reconocimiento de escenas o unas actuaciones que doten de cierta inteligencia al sistema implementado.

Una vez estudiada la estructura general de un sistema de internet de las cosas, se diseñó una infraestructura propia (Figura 4.2), basándose en la infraestructura general de IoT, con el objetivo de realizar el proyecto propuesto.

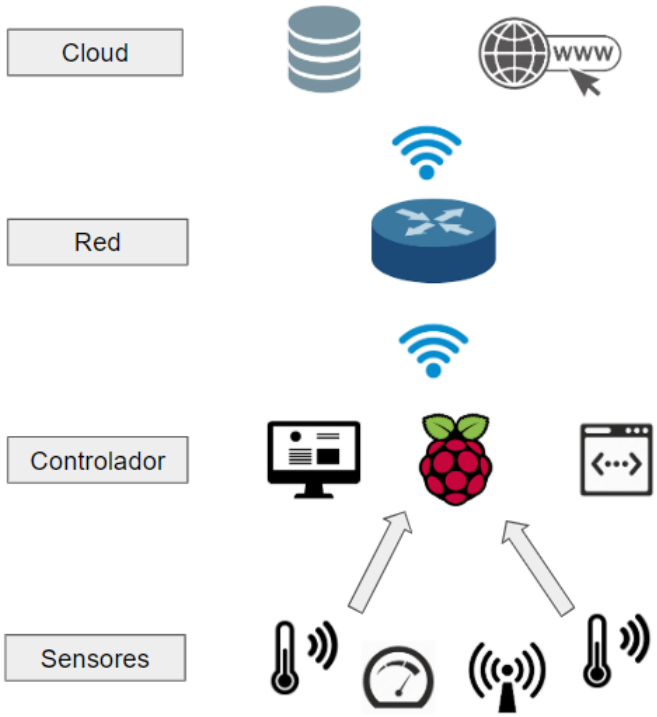


Figura 4.2: Infraestructura propia del sistema IoT implementado

Red de sensores: Se trata de un conjunto de sensores de diferentes proveedores. El objetivo principal de estos dispositivos es medir diferentes características físicas de la vivienda para obtener la mayor información posible de esta.

Controlador: El dispositivo Raspberry se incorpora al sistema con el objetivo de obtener la información generada por los sensores de la capa anterior. Esta información se obtiene haciendo uso de un protocolo de comunicación, fundamental para comunicarse con los sensores. Este controlador puede ejercer varias funciones, tanto en la niebla, como en la nube.

- **Niebla:** Haciendo uso de un software ya implementado, se pueden programar acciones en base a los datos proporcionados por los sensores.
- **Nube:** Se produce una inserción de los datos procedentes de los sensores en la base de datos de la nube para su posterior procesamiento.

Red: Mediante un router se realiza la conexión del controlador con los servicios que residen en la nube.

Nube: Se hace uso de servicios en la nube para almacenar, procesar y mostrar la información que procede del controlador y la información que se genera después del procesamiento de esta. Algunos servicios que se usan son:

- **Servicio de base de datos:** Se trata de un servicio que permite el alojamiento de bases de datos, accesibles a través de internet. Los datos que residen en el controlador, obtenidos a través de los sensores, se almacenan en este servicio.
- **Sistema web:** Mediante un sistema web implementado, se permite la visualización de los datos procedentes de los sensores y los datos generados por el análisis de estos datos.

A lo largo de las siguientes secciones, se muestra cómo se ha desarrollado e implementado el sistema IoT especificado, dividiéndolo en diferentes subsistemas con funcionalidad completa, hasta formar el sistema en sus totalidad.

5 Sistema IoT en la niebla o neblina

5.1 Infraestructura y objetivos

Una vez definida la estructura del sistema completo a desarrollar, se comenzó a realizar la parte que se encuentra a más bajo nivel. El sistema al final de este apartado consta de dos capas, la capa de dispositivos y la capa de perímetro, niebla o neblina. Como ya se ha visto en el apartado anterior, estas capas contienen los dispositivos físicos del sistema IoT, por lo que se trata de la parte más *hardware* del sistema. Así, la infraestructura desarrollada es la que se muestra en la Figura 5.1.

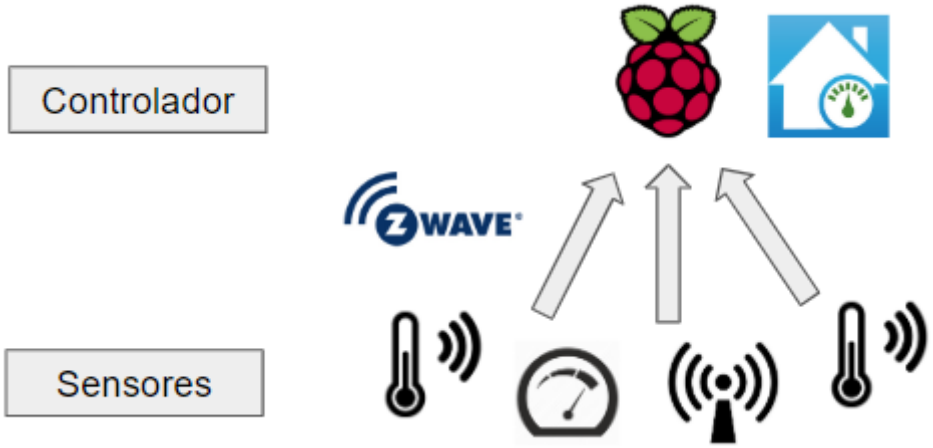


Figura 5.1: Equema del subsistema de computación en la niebla

Como se ha podido apreciar, se tiene una red de sensores, conectados a un controlador mediante un protocolo de comunicación. Una vez se encuentran los datos en el controlador, existe un *software* que permite la automatización de tareas con los datos proporcionados por los sensores.

Así, el resultado de la infraestructura desarrollada en este apartado consiste en un sistema de computación en la niebla, ya que los datos no cruzan la frontera del controlador. De esta forma, se pueden realizar las acciones básicas de un sistema IoT, como por ejemplo, obtener datos del entorno o realizar pequeñas acciones en base a estos datos.

En los siguientes apartados se desarrolla todo el proceso de implementación de esta infraestructura, incluyendo las tecnologías que se han usado y los pasos que se han conseguido para el funcionamiento de este.

5.2 Elementos del sistema

En este apartado se realiza una descripción de los elementos que forman parte del sistema.

5.2.1 Protocolo de comunicación

En primer lugar, fue necesario hacer un estudio sobre los diferentes protocolos de comunicación entre los sensores teniendo en cuenta los requisitos del sistema IoT a construir. Este estudio fue fundamental ya que a partir de la decisión que se adoptó en este momento, se condicionó toda la infraestructura a diseñar y construir, siendo diferentes los sensores que se usan para cada protocolo de comunicación, así como el controlador.

Como se ha mencionado, se tuvo que seleccionar un protocolo de comunicación entre los dispositivos que forman el sistema IoT en base a estas características generales:

- Se debe reducir al máximo el consumo de la batería que se emplea en el sistema.
- El sistema debe estar disponible durante todo el día.
- La comunicación del sistema tiene que ser inalámbrica.
- Los sensores pueden estar a una distancia máxima de 40 metros entre sí en interiores.

En base a estos requisitos, se estudiaron los protocolos de comunicación más usados para la domótica (Hersent y cols., 2011).

5.2.1.1 ZigBee

Se trata de un conjunto de protocolos para las comunicaciones inalámbricas, establecido por la ZigBee Alliance[1], y basado en el estándar *IEE 802.15.4*, con el objetivo de proporcionar conectividad a una gran cantidad de dispositivos, especialmente dispositivos de domótica, con el menor consumo energético posible.

Zigbee se trata de un estándar abierto, es decir, está disponible para las empresas con el objetivo de ser usado para implementar las comunicaciones entre los dispositivos que producen. Gracias a esto, está siendo popularizado últimamente, siendo cada día más los fabricantes que usan este estándar.

Por otra parte, este estándar es conocido por su bajo consumo de energía; esto se debe principalmente a que tiene una baja velocidad de transferencia de datos, con velocidades desde 40 kps hasta 250 kps.

Sin embargo, las distancias máximas entre dos dispositivos que se comunican no superan los 20 metros. Esto se debe a que los dispositivos que se implementan ZigBee se conectan formando una malla entre ellos y el controlador o *hub*. Este tipo de conexión consiste en que los dispositivos, para comunicarse con el *hub*, no realizan la conexión directa con este, sino que se transmiten

[1]Comunidad desarrolladora de estándares de software libre para redes inteligentes

la información entre ellos hasta alcanzar a este.

De esta forma, basta con tener los dispositivos cerca entre sí para que se pueda realizar una comunicación correcta. Esta conexión entre los dispositivos se facilita permitiendo la conexión de hasta 65.000 dispositivos en la misma malla de ZigBee. Además, la información se transmite entre los dispositivos hasta llegar al controlador, sin haber un número de saltos máximo.

En cuanto a la banda que usa para realizar las comunicaciones, se trata de la banda 868 MHz, 900 MHz o 2,4 GHz en Europa, la cual coincide con la banda del Wi-Fi. Esta compartición de banda entre ambas tecnologías puede provocar una saturación de la banda cuando se tiene un gran número de dispositivos conectados.

Por último, el protocolo de seguridad *AES-128* se usa en las comunicaciones entre los elementos que forman parte de la malla ZigBee.

5.2.1.2 ZWave

ZWave consiste en un protocolo desarrollado para permitir comunicaciones inalámbricas entre dispositivos. Este estándar fue diseñado con el objetivo de conectar diferentes dispositivos que facilitan la automatización del hogar, es decir, fue creado especialmente para el uso en domótica, por lo que se adapta a las necesidades de este ámbito.

Zwave se trata de un estándar cerrado, propiedad de Silicon Labs[2]. Al ser un estándar cerrado, se posee un mayor control sobre los dispositivos que usan este estándar, favoreciendo en cierto modo una mejora en la seguridad al utilizar este estándar.

Al igual que ZigBee, este estándar realiza la conexión de los dispositivos formando una malla entre ellos y el *hub* o controlador. De esta forma, para realizar la conexión entre un dispositivo y el *hub*, se tiene que transmitir esta información a otros dispositivos de la red para intentar llegar a este. Sin embargo, el número máximo de saltos en este caso es de 4. Esto hace que los dispositivos tengan que estar a una distancia no muy lejana con respecto al *hub*.

Si la distancia máxima en la que se pueden comunicar los dispositivos en este caso fuese de 10-20 metros como en el caso de ZigBee, este estándar sólo permitiría la comunicación en 40 metros. No obstante, este estándar permite la comunicación entre dispositivos distanciados hasta 100 metros.

En cuanto al número máximo de dispositivos permitidos en la malla Zwave, se permite un máximo de 232 dispositivos, siendo mucho menos que los 65.000 dispositivos aceptados con ZigBee.

Respecto a la velocidad de transferencia de datos, se tiene un rango de entre 9,6 y 100 kps, ya que las distancias son mayores. Así, en este caso, se hace uso de las bandas 868.40 MHz, 868.42 MHz y 869.85 MHz en Europa, por lo que no coincide con la banda del Wi-Fi, favoreciendo la

[2]Compañía dedicada a la creación de soluciones en el campo de IoT

independencia de esta comunicación respecto al Wi-Fi. Como consecuencia, en este caso no hay que preocuparse por las interferencias o sobrecarga en la banda usada.

5.2.1.3 Bluetooth

Bluetooth se trata de uno de los protocolos de comunicación inalámbrica más usados en la actualidad. Fue lanzado en el año 1994, basándose en el estándar *IEE 802.15.1*. Se trata uno de los protocolos de comunicación más utilizados para la conexión de dispositivos multimedia como altavoces, teclados, ratones, etc.

Esta comunicación con los dispositivos mencionados se puede realizar porque se tienen velocidades de transferencia de hasta 24 MB/s en el caso de tratarse de la versión *3.0*. Estas altas velocidades de transferencia hacen que los dispositivos tengan un mayor consumo de energía y por tanto una reducción del tiempo de la batería de estos dispositivos. Así, este protocolo se establece en una banda de frecuencia de 2,4 GHz, al igual que el Wi-Fi.

En cuanto a la topología de conexiones, se trata de una topología basada en el maestro-esclavo, en el cual el maestro mantiene una conexión con los esclavos, pero los esclavos no tienen una comunicación entre sí. De esta forma, en este protocolo se tiene un máximo de 8 dispositivos, siendo uno de ellos el maestro. Esta reducción del número de dispositivos es debida a la gran cantidad de información que se transmite en la comunicación maestro-esclavo, siendo necesarios bastantes recursos, por lo que no se pueden realizar un gran número de conexiones. En cuanto al alcance de las comunicaciones entre los esclavos y el maestro, la distancia es de hasta 30 metros.

5.2.1.4 Wi-Fi

Wi-Fi se trata de una tecnología inalámbrica cuyo objetivo es interconectar los dispositivos entre sí y con internet. Wi-Fi surgió a partir del estándar *IEE 802.11*, y en un principio trabajaba con ondas en las frecuencias de 2,4 GHz. Sin embargo, las mejoras que se están produciendo en la actualidad consisten en trabajar en otras frecuencias, como puede ser la frecuencia de 5 GHz.

Debido a la popularidad de esta tecnología, se crean constantemente numerosos estándares, como por ejemplo *IEE 802.11a*, *IEE 802.11b*, *IEE 802.11n*, etc.

En cuanto a las características principales de esta tecnología, se encuentra la distancia máxima de comunicación entre los dispositivos, siendo de hasta 200 metros en algunos estándares. Así, las velocidades de transferencia pueden superar los 300 MB/s.

Esta tecnología no está enfocada para ser usada en domótica, sino para realizar las comunicaciones entre los dispositivos de e internet, debido especialmente a su alto consumo de energía.

5.2.1.5 Comparativa y protocolo escogido

A continuación se muestra un cuadro-resumen comparativo de las características (Tabla 5.1) de los protocolos mencionados anteriormente.

Elemento	ZigBee	ZWave	Bluetooth	Wi-Fi
Banda Frecuencia	2,4 GHz	868 MHz	2400 GHz	2,4 GHz o 5 GHz
Velocidad	40-250 KB/s	9,6-100 KB/s	24 MB/s	hasta 300 GB/s
Distancia	hasta 10-20 m	hasta 100 m	hasta 30-40 m	hasta 200 m
Num. Max. dispositivos	65.000	232	8	depende red
Precio medio	Bajo	Bajo	Medio	Alto
Consumo	Bajo	Bajo	Medio	Alto

Tabla 5.1: Comparativa de características de protocolos de comunicación

En vista de las características de los protocolos de comunicación inalámbrica que se han estudiado anteriormente, la domótica en estos momentos está centrada en el uso de los dos primeros protocolos: ZigBee y ZWave, por lo que se va a decidir usar uno de ellos (Lee y cols., 2007).

En base a las necesidades del sistema enumeradas anteriormente, se ha optado por el uso del protocolo ZWave, ya que permite la conexión de hasta 232 sensores y en la vivienda se usan un máximo de 20, a distancias superiores a 10-20 metros entre sí y entre el controlador, por lo que no hay problema en establecer la comunicación entre estos y el controlador. Además, dada su baja velocidad en la transferencia de archivos, se tiene un menor uso de recursos, permitiendo una larga vida a la batería de los sensores.

Se podría haber optado por ZigBee, pero no se van a querer conectar una gran cantidad de sensores. Además, la distancia entre estos tiene que ser menor (no es suficiente un máximo de 20 metros) y la velocidad de transferencia de los datos no es un aspecto crucial, por lo que en este caso, se adapta más el sistema al protocolo ZWave.

5.2.2 Sensores

Según (de Valladolid, 2019), "Un sensor es un dispositivo eléctrico y/o mecánico que convierte magnitudes físicas (luz, magnetismo, presión, etc.) en valores medibles de dicha magnitud. Este proceso sigue tres fases. En primer lugar, se capta un fenómeno físico medible por el sensor y se genera una señal eléctrica dependiente del valor de la variable física. En segundo lugar, la señal eléctrica es modificada por un sistema de acondicionamiento de señal cuya salida es voltaje. En último lugar, el sensor dispone de una circuitería que transforma y/o amplifica la tensión de salida, la cuál pasa a un conversor A/D, conectado a un PC. El convertidor A/D transforma la señal de tensión continua en una señal discreta."

En la actualidad, los sensores cada vez son más pequeños, por lo que se incorporan a una gran cantidad de dispositivos como smartphones, coches, etc (Hanes y cols., 2017).

5.2.2.1 Tipos de sensores

Existe una gran cantidad de sensores creados para medir las propiedades físicas de la naturaleza, con objetivo de obtener información de estas para poder usarlas con algún beneficio. Así, a continuación se exponen los principales sensores que se comercian actualmente en el campo de IoT, junto con un ejemplo de uso y las magnitudes físicas que miden (Hanes y cols., 2017).

Sensor de presión: Este sensor se usa para detectar cuándo se presiona una superficie. A través de este sensor se puede detectar por ejemplo cuándo una persona está durmiendo en la cama, y calcular el tiempo total que esta persona ha estado durmiendo.

Sensor de luminosidad: Se trata de un sensor que mide la intensidad lumínica que recibe el fotorreceptor, y por lo tanto permite recabar información sobre la luz que hay en el ambiente donde esté instalado. Esta información lumínica permitirá realizar actuaciones tales como subir o bajar persianas, o bien encender o apagar luces.

Acelerómetro: Un acelerómetro es un dispositivo que mide la vibración o la aceleración del movimiento de una estructura, generando una carga eléctrica que es proporcional a la fuerza que provocó el movimiento o vibración. De esta forma, cuando se mueve este sensor, se obtendrá cierta señal, dependiendo de estos parámetros físicos que se están midiendo. De esta forma, cuando se mueve la estructura sobre la que está instalado el sensor, se obtendrá una señal indicativa de la aceleración del movimiento. Un ejemplo de uso es el acelerómetro que llevan incorporados los teléfonos móviles que pueden detectar una posible caída de dicho dispositivo y se apaga el móvil antes de que impacte contra el suelo.

Sensor de humedad: Este sensor monitoriza la humedad del ambiente, permitiendo realizar diferentes acciones con los resultados que aporta, como por ejemplo, activar el riego en los cultivos cuando la humedad sea inferior a un valor establecido.

Sensor de contacto: Es un tipo de sensor que mide la posición de un objeto haciendo contacto directamente con él. Los sensores de contacto son los más simples y básicos, se activan en el momento en que entran en contacto con un objeto. Mayoritariamente, se emplean para detectar el final del recorrido o la posición límite de componentes mecánicos. Por ejemplo, se puede usar para saber si se abre o se cierra una ventana.

Sensor de temperatura: En este caso, la magnitud que se va a medir se trata de la temperatura del ambiente. Este sensor puede ser usado para activar la calefacción cuando la temperatura es menor a una temperatura establecida.

Sensor ultrasónico: Este sensor emite un sonido y mide el tiempo que la señal que se ha emitido tarda en regresar. De esta forma, permite la medición de la proximidad del sensor respecto al objeto que se detecta. Este sensor se puede usar para detectar presencia de una persona en una habitación. Una posible actuación podría ser la de encender la luz automáticamente al detectar dicha presencia.

5.2.2.2 Sensores a utilizar

Los sensores que se van a usar para la creación del ambiente inteligente se tratan de sensores inalámbricos, es decir, cuya transmisión de datos hacia el controlador no se realiza mediante cable, sino por el ambiente. Concretamente, se ha optado por el uso de sensores que implementen el protocolo de comunicación Zwave, comentado con anterioridad.

Además, existen dispositivos en el mercado que incluyen diferentes tipos de sensores encapsulados. Esto puede ser útil ya que la cantidad de dispositivos a instalar en el ambiente inteligente

se reduce en gran parte.

5.2.2.2.1 Motion Sensor FIBARO Este dispositivo es un multisensor de fácil instalación, que funciona a través del protocolo ZWave. Al tratarse de un multisensor, contiene los siguientes sensores:

- Sensor de luminosidad
- Sensor de temperatura
- Sensor de presencia
- Acelerómetro

Figura 5.2: Motion Sensor Fibaro

En cuanto al sensor de presencia, este dispositivo usa un sensor *IR pasivo*. Este sensor de infrarrojos pasivo se encarga de diferenciar el calor emitido por fuentes de energía como el calor humano por el emitido por objetos estáticos e inertes. Así, de esta forma se puede incorporar el sensor en cualquier espacio interior, aunque haya objetos en el rango de detección, ya que estos no emiten el calor que emite el humano, por lo que no detectaría la presencia de estos objetos.

Además, contiene un *LED* que se se enciende y modifica su color cuando se produce algún cambio en las magnitudes que miden los sensores incorporados.

Especificaciones técnicas Las especificaciones técnicas (Fibaro, 2019c) del sensor *Motion Sensor FIBARO* se especifican en la Tabla 5.2.

Configuración manual Cada sensor, tiene una configuración por defecto, pudiendo ajustar diferentes parámetros para adaptarlo a las necesidades del usuario final; así, para el uso de este sensor, se ha realizado un cambio sobre la configuración inicial.

- Retraso de cancelación de alarma: Se trata del tiempo que permanece activo el sensor una vez que ha detectado movimiento. La configuración predeterminada es de 30 segundos, pero en este trabajo se ha incrementado a 60 segundos para tener más posibilidades de seguir

Característica	Valor
Rango de luminosidad	0 - 32.000 LUX
Rango de error temperatura	0.5 ºC
Distancia	40 m
Frecuencia de radio	868,4 u 869,8 MHz UE
Potencia de señal de radio	hasta -5 dBm
Dimensiones (Largo,Alto,Ancho):	47.7 x 46 x 45.5 mm

Tabla 5.2: Especificaciones técnicas Motion Sensor Fibaro

detectando presencia en el caso de que la persona haga pequeños movimientos que no son capaces de activar el sensor de movimiento tras el estado de desactivado. Hay que indicar que cuando el sensor detecta movimiento, se desactiva durante los 10 primeros segundos de esa ventana, quedando así 50 segundos (y no 20) para detectar posibles movimientos.

Uso El dispositivo se configuró principalmente para actuar como sensor de detección de presencia. En ciertas habitaciones este sensor tiene especial importancia ya que se tienen en cuenta los datos de los sensores cuando este sensor está activo. Así, se puede calcular cuánto tiempo exactamente ha estado una persona en la zona de detección de este sensor.

5.2.2.2.2 Door/Window Sensor 2 Al igual que el sensor *Door/Window Sensor*, se trata de un dispositivo (Figura 5.3) que engloba varios sensores en un único dispositivo. En este caso contiene los siguientes sensores integrados:

- Sensor de temperatura
- Sensor de contacto
- Acelerómetro

Figura 5.3: Door/Window Sensor 2

A diferencia del sensor anterior, *Door/Window Sensor 2* no permite una conexión externa, ni de conectores *switch* ni de temperatura; sin embargo, integra un sensor de temperatura nativo, como hacen otros dispositivos Fibaro como el caso de *Motion Sensor FIBARO*, explicado anteriormente.

Especificaciones técnicas Las principales especificaciones técnicas (Fibaro, 2019b) del sensor *Door/Window Sensor 2* se muestran en la Tabla 5.3.

Característica	Valor
Temperatura de funcionamiento	0 ºC - 40 ºC
Rango de temperatura medible	0 ºC - 60 ºC
Error en temperatura	0,5 ºC
Distancia	40 m
Frecuencia de radio	868,4 MHz UE
Potencia de señal de radio	hasta -5 dBm
Dimensiones (Largo,Alto,Ancho):	71 x 18 x 18 mm

Tabla 5.3: Especificaciones técnicas Door/Window Sensor 2

Configuración manual En algunos sensores ha sido necesario realizar algún cambio en la configuración por defecto de estos. Así, la única propiedad que se ha cambiado ha sido la siguiente:

- Estado de la puerta / ventana. La configuración por defecto es la mostrada en la Tabla 5.4.

Estado físico	Valor	Estado sensor
Ambas partes juntas	0	OFF
Ambas partes separadas	1	ON

Tabla 5.4: Estados posibles Door/Window Sensor 2

En algunos casos, ha sido necesario invertir estos valores, ya que cuando ambas partes están separadas es cuando el objeto sobre el cual está situado el sensor está cerrado, en lugar de abierto.

Como resultado de la modificación de los valores por defecto, se tiene el comportamiento que se muestra en la Tabla 5.5:

Estado físico	Valor	Estado sensor
Ambas partes juntas	1	ON
Ambas partes separadas	0	OFF

Tabla 5.5: Estados posibles Door/Window Sensor 2 - Configuración Modificada

Uso Este dispositivo se ha usado principalmente como sensor de contacto, establecido en diferentes puertas para detectar cuándo se abren y cuando se cierran. En cuanto al sensor de temperatura, aunque mide la temperatura de forma correcta, en las habitaciones en las que

existen sensores *Motion Sensor FIBARO* y *Window/Door Sensor 2*, se ha optado por hacer uso de los valores de la temperatura que proporciona el sensor *Motion Sensor FIBARO*, ya que tiene un rango de detección de temperatura mayor.

5.2.2.2.3 Arun PM3 Este sensor se trata de un dispositivo (Figura 5.4) diferente a los anteriormente mencionados, ya que la compañía que lo ha creado no se trata de Fibaro; así, no se trata de un multisensor, sino que se trata de un sensor de presión binario, con solo dos estados posibles, encargado de detectar presión sobre su superficie.

Figura 5.4: Sensor de presión

En el circuito interno del sensor (Figura 5.5) se pueden observar diferentes circuitos:

- Cerrado (izquierda): Tamper loop: Se trata de la conexión a una alarma, cuando se presiona, se dispara la alarma.
- Abierto (derecha): Se trata de la conexión que se realiza para detectar presión; tiene un estado de normalmente abierto.

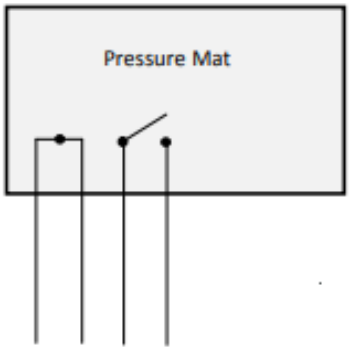


Figura 5.5: Circuitos sensor Arun PM3

Como se ha comentado, se trata de un sensor binario, por lo que solamente tiene dos estados posibles. Estos estados y su correspondencia al estado real del sensor se pueden apreciar en la

Tabla 5.6

Valor	Estado
0	Sin presionar
1	Presionado

Tabla 5.6: Estados posibles Sensor de presión

Especificaciones técnicas Las principales especificaciones técnicas (Electronics, 2019) del sensor DArun PM3 se muestran en la Tabla 5.7

Característica	Valor
Estado natural	Normalmente abierto
Rango de temperatura	-10 - 70 ºC
Mínima presión detectada	25 kg en 50 mm
Maximo voltaje	25 Vcc
Dimensiones (Largo,Alto,Ancho):	720 x 560 x 3 mm

Tabla 5.7: Especificaciones técnicas Arun PM3

Uso Este sensor de presión se ha usado para comprobar cuándo una persona se sienta sobre una superficie. Dado que la conexión de este sensor se realiza a través de cables, para comunicarse con el controlador habría que situar un controlador cerca de cada sensor de presión. Para evitar esta conexión directa, se hace uso de un dispositivo intermediario. Este dispositivo se trata del sensor de contacto Door/Window Sensor descrito en 5.2.2.2.4. De esta forma, el sensor de contacto recibe los cambios de estado del sensor de presión a través del cableado que se muestra en (Figura 5.6) y desde este sensor de contacto se envía la señal al controlador haciendo uso del protocolo ZWave.

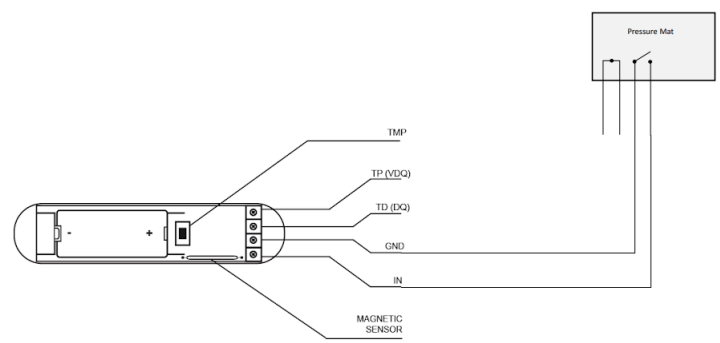


Figura 5.6: Conexión Arun PM3 - Door/Window Sensor

5.2.2.2.4 Door/Window Sensor Al igual que el sensor *Motion sensor FIBARO*, se trata de un dispositivo de la compañía Fibaro. Este multisensor (Figura 5.7) contiene los siguientes sensores incorporados:

- Acelerómetro
- Sensor de contacto
- Conexión externa

Figura 5.7: Door/Window Sensor

En cuanto al sensor de contacto, este dispositivo está formado por dos piezas, siendo una de ellas una pieza magnética y la otra pieza el cuerpo del sensor. Cada vez que se juntan o separan estas piezas, se produce un cambio en el estado del sensor. Así, la distancia máxima para la detección de la parte magnética del sensor es de 5 mm (Figura 5.8).

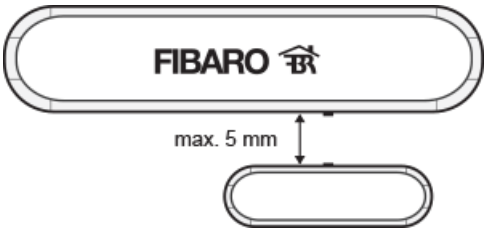


Figura 5.8: Distancia partes Door/Window Sensor

Por otra parte, hay que destacar la presencia de varios pines (Figura 5.9) en los cuales se pueden instalar diferentes dispositivos. Los pines son los siguientes:

- TP: Temperature Power: En este pin se proporciona potencia para el sensor de temperatura que se conecte.
- TD: En este pin se recogen los datos del sensor conectado.

- GND: Se trata del pin de tierra.
- IN: Se trata de un pin libre de potencial, al cual pueden conectarse elementos *switch*.

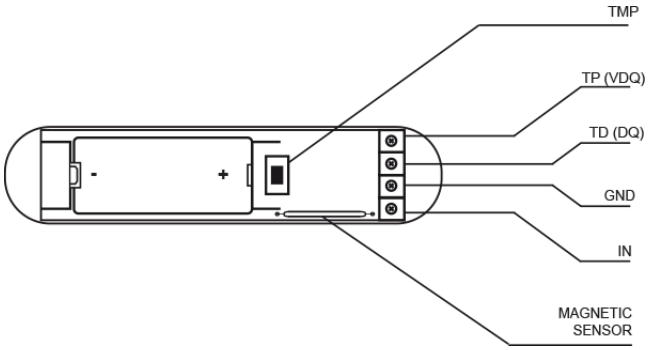


Figura 5.9: Conexión externa Door/Window Sensor

Especificaciones técnicas Las principales especificaciones técnicas (Fibaro, 2019a) del sensor Door/Window Sensor se muestran en la Tabla 5.8

Característica	Valor
Temperatura de funcionamiento	0 ºC - 40 ºC
Distancia	40 m
Frecuencia de radio	868.4 MHz UE
Potencia de señal de radio	hasta -5dBm
Dimensiones (Largo,Alto,Ancho):	76 x 19 x 17 mm

Tabla 5.8: Especificaciones técnicas Door/Window Sensor

Uso En lugar de utilizarse como sensor de contacto se ha utilizado como intermediario entre el controlador y el sensor de presión al carecer este último de la posibilidad de comunicación inalámbrica con el controlador. El sensor de presión consta de tres cables, dos de ellos forman un circuito abierto que, cuando se mantiene presionado, se cierra el circuito y esto significa que el sensor está presionado. Para conectar estos cables con el controlador, se puede hacer a través de los puertos GPIO; sin embargo, se necesitarían tener varios controladores conectados para recoger los datos de los diferentes sensores de presión y supondría un coste más elevado.

Esto se soluciona usando este sensor, ya que las dos salidas del sensor de presión se colocan en los pines *IN* y *GND* de este sensor y este notifica al controlador cuando se presione, por lo que la conexión del sensor de presión y el *hub* es más fácil e inalámbrica.

5.2.2.3 Localización

La vivienda en la cual se han instalado los sensores se trata de un piso, situado en El Ejido (Almería). En este alojamiento actualmente residen 4 personas, por lo que se distribuyen los

sensores en base a los lugares en los cuales se encuentra más a menudo el usuario que gestiona el sistema.

- Cuarto de baño
  - Door/Window Sensor 2 (Sensor de contacto)
    - WC
    - Puerta de entrada
  - Motion Sensor Fibaro (Sensor de presencia)
    - Interior del cuarto de baño
    - Interior de la ducha

    Estos sensores de presencia se han dispuesto de forma que el sensor que se coloca en el interior de la ducha solamente detecta presencia cuando una persona está dentro de la ducha. Así, la disposición de los sensores, especialmente de presencia, variará dependiendo de cada vivienda.

- Cocina
  - Door/Window Sensor 2 (Sensor de contacto)
    - Microondas
    - Frigorífico
    - Cajón de los cubiertos
    - Puerta de la vajilla
    - Puerta de la comida
    - Puerta de los utensilios para cocinar
  - Motion Sensor Fibaro (Sensor de presencia)
    - Entrada de la cocina
    - Salida a la terraza

    Estos sensores de presencia se han colocado en la cocina, con el objetivo de abarcar el máximo volumen a detectar, para definir bien las áreas de la cocina que se usan cuando se realizan las diferentes actividades en ella.

- Salón
  - Door/Window Sensor (Sensor de contacto, intermediario) + Sensor de presión
    - Sofá 1 parte izquierda
    - Sofá 1 parte derecha
    - Sofá 2 parte izquierda
    - Sofá 2 parte derecha

    Se han dispuesto dos sensores en cada sillón ya que es el número máximo de personas que pueden ocupar el mismo. De esta forma, se pueden detectar hasta 4 personas en total entre los dos sofás que tiene la vivienda.

- Terraza
  - Door/Window Sensor 2 (Sensor de contacto)
    - Lavadora
- Dormitorio
  - Door/Window Sensor (Sensor de contacto, intermediario) + Sensor de presión
    - Cama parte superior
    - Cama parte inferior

    Se han dispuesto dos sensores de presión en una cama para poder diferenciar cuándo una persona se está vistiendo o simplemente está durmiendo.

De esta forma, la vivienda equipada con todos los sensores mencionados quedaría de la forma que se muestra en la Figura 5.10. En la imagen, se diferencian los tres tipos de sensores que se han instalado; sensores de movimiento, de contacto y de presión.

Figura 5.10: Ubicación de los sensores en la vivienda

La distribución de los sensores en las habitaciones se ha basado en las actividades básicas que realizan los usuarios en la vivienda (Ghayvat y Mukhopadhyay, 2017). Sin embargo, en la bibliografía citada, se incluye una numerosa cantidad de sensores (Figura 5.11), lo cual no es aplicable en el caso del trabajo actual.

5.2.3 Controlador

Según Cisco (2018-2019), "Los controladores son responsables de recopilar datos de los sensores y proporcionar conectividad hacia la red o Internet. Los controladores pueden tener la

Activities	Sensor type	Installation location
Prepare food	PIR, light, temperature	Kitchen desk
	Contact	Cabinets, fridge doors, kitchen door
	Gas, smoke	Kitchen ceiling
	E and E unit	Plugged with rice cooker, oven, toaster, water kettle, microwave oven
Washing dishes	PIR	Kitchen window, appliances
	Contact	Cabinets
Having meal	PIR	Kitchen corner
	Pressure	Dining chair
Watching TV	PIR	Living area
	Pressure	Chairs, sofa
	E and E unit	TV plug
Toilet	PIR	Corner of bathroom
	Pressure	Toilet seat
Shower	Contact	Shower door
Get up	PIR	Corner of bedroom
	Pressure	Below the bed
Go to bed	PIR	Corner of bedroom
	Pressure	Below the bed
All indoor activities	Accelerometer	Neck
Relax	Pressure	Chair

Figura 5.11: Caso de ejemplo de actividades y sensores

capacidad de tomar decisiones inmediatas o de enviar datos a una computadora más potente para su análisis." Así, estos dispositivos tienen que tener la capacidad de ser el enlace entre los datos que se aportan mediante los sensores y el tratamiento que se hace con estos datos.

En la actualidad, se están usando como controladores dispositivos de pequeño tamaño pero con capacidad de computación, de tal forma que estos puedan procesar los datos que se reciben y reenviarlos a la nube o bien, realizar algunas acciones en base a estos.

Los dispositivos que más se están usando como controladores para proyectos IoT en la actualidad se tratan de Arduino y Raspberry. En este caso, se ha decidido hacer uso de Raspberry, ya que se trata de un *PC* a pequeña escala, a diferencia de Arduino, que consiste en una placa en la cual se puede manejar la electrónica que se conecte a él mediante la programación de pequeñas aplicaciones. Además, permite la conexión a Internet mediante Wi-Fi, s de pequeño tamaño, y contiene un conjunto de entradas/salidas para conectar dispositivos, por lo que hace de este dispositivo un dispositivo idóneo para proyectos IoT (Rao, 2018).

5.2.3.1 Raspberry Pi 3 B+

Raspberry Pi 3 B+ (Figura 5.12) es un producto que forma parte de la familia de dispositivos Raspberry. Estos dispositivos son especialmente conocidos gracias a su amplia funcionalidad (tiene las mismas funciones que un *PC*), a muy bajo precio (sobre 35 euros). Además, gracias a los pines *GPIO* de los que dispone, se permite la conexión de sensores o elementos electrónicos, lo cual hacen que este producto pueda usarse para la programación de sistemas de igual modo a como se hace con Arduino.

Figura 5.12: Raspberry Pi 3 B+

Tipo de dispositivo Este dispositivo se trata de un SBC[3], esto es, una placa que actúa como un PC normal, ya que tiene los elementos fundamentales de este, tales como memoria RAM, puertos de entrada/salida y microprocesador. Concretamente, este dispositivo además de estos elementos básicos, contiene los siguientes añadidos:

- Puerto USB: Permite la conexión mediante USB de otros dispositivos.
- Puerto HDMI: Permite la conexión de una pantalla para visualizar el contenido.
- Puerto Ethernet: Permite la conexión a la red.
- Pines GPIO: Permiten la conexión de dispositivos o sensores en estos pines digitales.
- Conector para cámara: Permite la conexión de una cámara al SBC.
- Memoria externa: Permite la conexión de una tarjeta microSD.
- Conector de alimentación: Permite proporcionar energía al dispositivo para su funcionamiento.

[3]Single Board Computer (Computadora de una sola placa)

- Conexiones inalámbricas: Permite el uso de Wi-Fi y Bluetooth.

En el caso de Raspberry Pi, el microprocesador se encuentra integrado en un SoC[4]. Un SoC se trata de una placa que integra los elementos básicos de un PC, tales como la CPU, la memoria caché, la tarjeta gráfica integrada, los buses, y la conexión con el resto de puertos del sistema. Este sistema se encuentra en una placa de pequeñas dimensiones, que es cada vez más usada en dispositivos móviles y SBCs.

El SoC que está presente en Raspberry Pi 3 B+, concretamente el modelo *Broadcom BCM2837B0 Cortex-A53*, contiene los siguientes elementos:

- CPU
- GPU
- Memoria caché
  - Nivel 1
  - Nivel 2
- DSP[5]
- Puerto USB

En cuanto a los pines GPIO, el esquema y la distribución de los mismos se puede observar en la Figura 5.13.

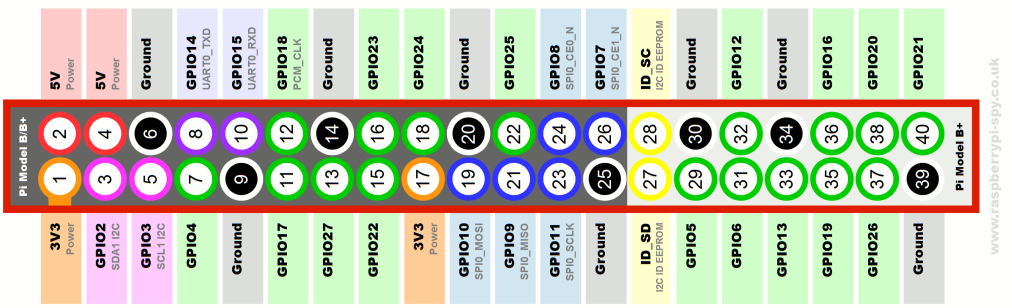


Figura 5.13: Esquema pines GPIO

Características hardware Una vez se han enumerado todos los componentes que forman parte de Raspberry Pi 3 B+, se muestran las especificaciones técnicas de estos elementos (Foundation, 2019).

- SoC: Broadcom BCM2837
  - CPU: ARMv8, 4 núcleos, 1.4GHz

[4]System On a Chip
[5]Procesador digital de señales

  - GPU: Dual Core VideoCoreIV, 400MHz
  - SDRAM
    * 32kB nivel 1
    * 512kB nivel 2
  - Procesador digital de señales
  - Puerto USB
- Memoria: 1GB LPDDR2 SDRAM
- Conectividad:
  - Puertos USB: 4xUSB 2.0
  - Puerto Ethernet: GigabitEthernet
  - Conexión Wi-Fi: IEE 802.11.g/g/n/ac, con frecuencia de 2,4GHz y 5GHz
  - Conexión Bluetooth: 4.2
- Pines GPIO: 40 pines
- Vídeo y sonido:
  - Puerto HDMI
  - Puerto MIPI DSI (video)
  - Puerto MIPI CSI (cámara)
- Puerto para insertar tarjeta microSD
- Entrada de energía
  - 5V/2.5A a partir del puerto USB
  - 5V DC a través de puertos GPIO

Sistema Operativo Raspberry Pi 3 B+ al tratarse de un ordenador, puede ser configurado para tener el sistema operativo que el usuario desee. Así, se ha diseñado un sistema operativo específico para usar en los diferentes modelos de Raspberry.

Este sistema operativo se trata de *Raspbian*, que es software libre, creado por un grupo de desarrolladores que no pertenecen a la Fundación Raspberry Pi. Este software está basado en *Debian*, y ha sido optimizado para el *hardware* del dispositivo Raspberry. De esta forma, una vez se instala el sistema operativo *Raspbian*, se tiene un conjunto de paquetes que facilita el uso del sistema, realizando la configuración de los parámetros básicos del *PC*.

Si un usuario no desea instalar este sistema operativo, puede instalar cualquier otro sistema operativo. Algunos ejemplos de sistemas operativos son los siguientes, como pueden ser los sistemas operativos incluídos en *NOOBS*[6].

- LibreELEC

[6]Instalador de versiones de sistemas operativos compatibles con Raspberry

- Raspbian Lite
- Lakka
- Arch
- OpenELEC
- Pidora
- RISC OS

Configuración inicial Una vez se estudió tanto el *software* como el *hardware* del dispositivo Raspberry Pi 3 B+, se procedió a la instalación del sistema operativo e inicio del sistema (Schwartz, 2016). El sistema operativo instalado se trata de *Raspbian*, concretamente la versión XXXXX. Para instalar el sistema operativo, se realizaron los siguientes pasos:

- Descarga del software Raspbian *v9.0*
- Descompresión del archivo haciendo uso de *WinRAR*
- Formateo de la tarjeta microSD
- Copia del archivo que contiene el sistema operativo en la tarjeta microSD.
- Inserción de tarjeta microSD en el puerto correspondiente en el dispositivo Raspberry Pi 3B+.

Una vez se ha instalado el sistema operativo en la tarjeta microSD, se procedió al inicio del sistema (Figura 5.14).

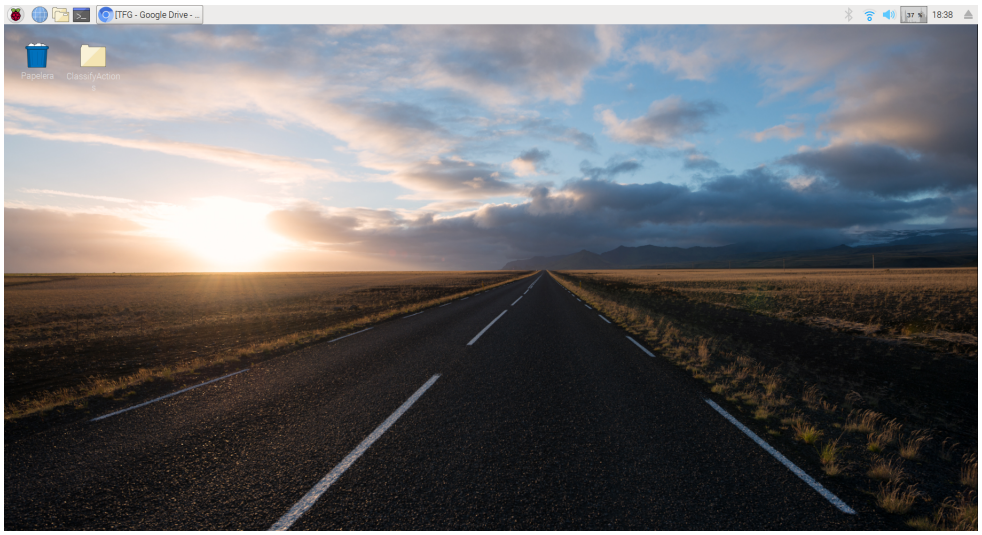


Figura 5.14: Escritorio Raspbian

Una vez se inició el equipo, se tuvieron que realizar algunos cambios en la configuración predeterminada.

- Activación de acceso mediante SSH[7].

  Al establecer el acceso mediante SSH, se permite acceder a Raspberry Pi 3 B+ desde otro dispositivo, que en este caso se trata del ordenador personal. La conexión se ha realizado para cargar scripts o realizar modificaciones en la configuración.

  De esta forma, no es necesario conectar una pantalla a Raspberry Pi 3 B+ para realizar configuraciones sobre esta.

- Login automático cuando se inicia.

- Inserción de contraseña del Wi-Fi.

Cuando se inició y configuró el dispositivo Raspberry correctamente, se realizó la integración del dispositivo que va a permitir la conexión de Raspberry con los sensores que se han mencionado anteriormente, es decir, el dispositivo que va a convertir a Raspberry Pi en un controlador en un ambiente de IoT.

5.2.3.2 Módulo Razberry

Actualmente, hay bastantes compañías que aportan soluciones de conectividad entre sensores y controladores mediante el uso del protocolo ZWave. Dado que en este caso se ha decidido usar este protocolo, se han buscado controladores y sensores que se comuniquen haciendo uso de este.

En particular se ha elegido el dispositivo Razberry, que ha sido desarrollado e implementado por la compañía ZWave.me. Dicha compañía fue fundada por un grupo de ingenieros que querían implementar una red basada en el protocolo ZWave y que facilitase la creación de entornos inteligentes basados en sensores y actuadores.

El dispositivo Razberry es un módulo (Figura 5.15) que se puede acoplar a cualquier versión del dispositivo Raspberry y que permite la creación de un sistema de control de automatización de un ambiente inteligente basado en el protocolo Zwave.

La principal función que tiene este módulo consiste en realizar la conexión e intercambio de información entre los sensores del ambiente inteligente y la Raspberry, es decir, actúa acoplado a la Raspberry y la transforma en un controlador de IoT.

Características hardware Las principales características hardware del dispositivo Razberry se tratan de las mostradas en la Tabla 5.9

Característica	Valor
Tipo de módulo	Controlador Z-Wave Plus
Chipset	Sigma Designs ZM5202 (5ª generación)
Fuente de alimentación	Puerto GPIO
Frecuencia	868.42 MHz
Alcance	hasta 40 m en interiores.
Dimensiones	20 mm x 40 mm x 3 mm

Tabla 5.9: Características hardware de Razberry

[7]Secure SHell

Figura 5.15: Módulo Razberry

Conexión con Raspberry El módulo Razberry se conecta con Raspberry a través de los puertos GPIO de esta (Figura 5.16). Así, una vez se conecta a estos puertos, está listo para usarse.

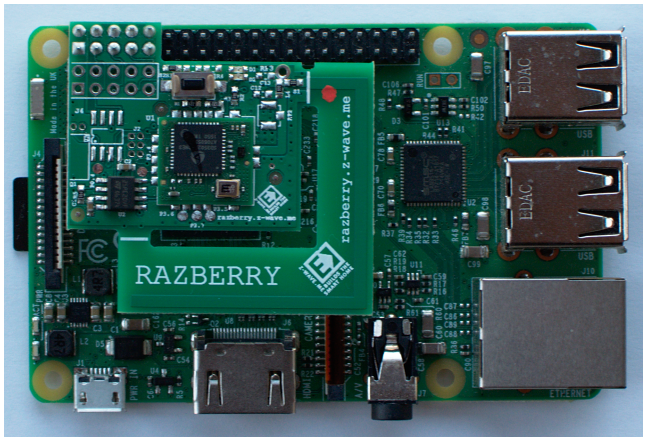


Figura 5.16: Conexión Razberry y Raspberry Pi 3 B+

Software Z-Way El módulo Razberry convierte la Raspberry en un controlador de IoT, permitiendo su comunicación con sensores que implementen el protocolo Zwave. Sin embargo, para poder manejar esta comunicación, se tiene que proporcionar un software que maneje esta. Hay distintos proveedores que han desarrollado software para módulos basados en Zwave. Algunas aplicaciones software que se pueden usar pueden ser *Domoticz*, *Jeedom* o *Z-Way* entre otros.

Así, junto con el módulo físico, se proporciona un software llamado Z-Way. Este software,

convierte a la Raspberry en un centro de monitorización del ambiente inteligente, proporcionando las siguientes funcionalidades principales:

- Adición de nuevos dispositivos.
- Monitorización del estado de los sensores.
- Programación de tareas en base a los datos de los sensores.
- Configuración de los parámetros de los sensores.
- Integración de los datos con otras aplicaciones de terceros.

Este software se ejecuta en Raspberry, siendo necesaria la **instalación** del mismo en el dispositivo. Una vez que se tiene instalado el sistema operativo en la Raspberry, se le ha conectado el módulo Razberry y se tiene conexión a internet, el software Z-Way se instala siguiendo los siguientes pasos:

- Ejecución del siguiente comando

Código 5.1: Instalación Z-Way

```
1        wget -q -O - razberry.z-wave.me/install | sudo bash
```

- Acceso a la interfaz web en la cual se muestra el software
- Introducción de usuario y contraseña

Una vez se han realizado estos pasos, se visualiza una pantalla en la cual se muestran los sensores que se han emparejado correctamente con el dispositivo (Figura 5.17).

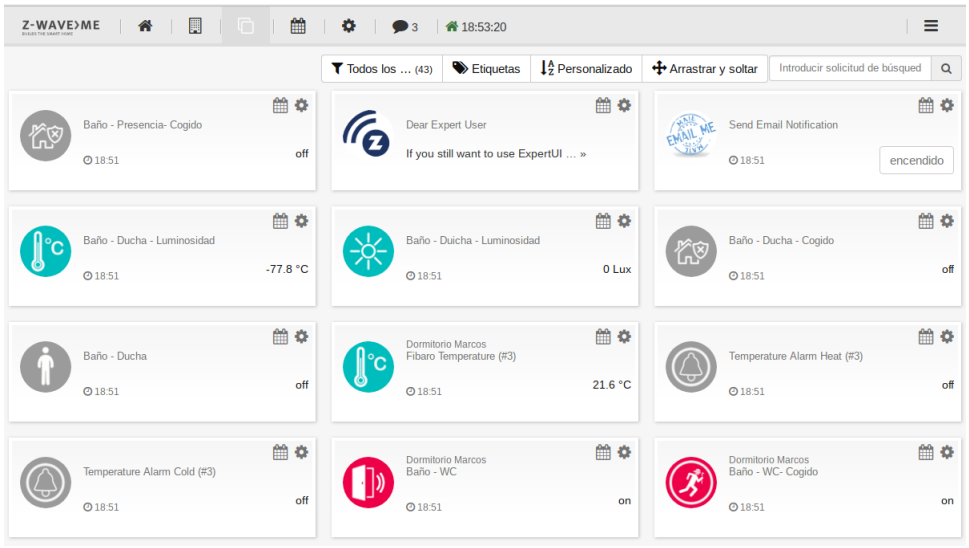


Figura 5.17: Pantalla principal Z-Way

Este software, como se ha mencionado anteriormente, permite una amplia variedad de **funcionalidades**; sin embargo, en este proyecto, solamente se usa para incorporar nuevos dispositivos

al sistema y modificar sus parámetros. A continuación se comentan los pasos a seguir para la realización de estas.

- Incorporación nuevo dispositivo: Para realizar la incorporación de un nuevo sensor al sistema (Figura 5.18), se siguieron los siguientes pasos:
  - Activación el sensor
  - Activación del modo compatibilidad del sensor
  - Detectar nuevo dispositivo en la interfaz web
  - Establecimiento de nombre

Figura 5.18: Adición de dispositivo a la red ZWave

- Modificación configuración de dispositivo: En ciertos casos, fue necesario cambiar la configuración por defecto de los sensores a incluir en el sistema (Figura 5.19). Para ello, se siguieron los siguientes pasos:
  - Visualización de los sensores
  - Edición sensor seleccionado
  - Configuración del hardware
  - Activar modo compatibilidad del sensor
  - Cambiar configuración
  - Guardar cambios

5.3 Presupuesto

A continuación en la Tabla 5.10 se muestra la cantidad de dispositivos que han sido necesarios para la construcción del sistema de IoT en la niebla que se ha desarrollado en este apartado, así

Figura 5.19: Cambio configuración parámetros

como el coste final de la infraestructura.

Elemento	Precio (€)	Cantidad	Total (€)
Motion Sensor FIBARO	49	4	196
Door/Window Sensor	51	6	306
Door/Window Sensor 2	38	9	342
Razberry	59	1	59
Raspberry Pi 3 B+	60	1	60
Arun PM3	12	6	72
Total	**-**	**27**	**1.035**

Tabla 5.10: Presupuesto

5.4 Resultado

Tal y como se comentó en la introducción de esta sección, se ha construído un sistema de computación en la niebla, haciendo uso de diferentes sensores, un controlador y un software para manejar los datos de estos sensores. Sin embargo, como se ha podido observar, no se obtiene todo el potencial de IoT simplemente teniendo este sistema. Se trata de un sistema en el cual los datos de los sensores no se pueden modificar al gusto de la persona que trabaje con esta tecnología, y no se pueden analizar estos datos usando gráficas o similares. Por todos estos motivos, nace la necesidad de extraer estos datos con una aplicación o sistema personalizado, capaz de usar estos datos generados por los sensores de la forma que se quiera.

En la siguiente sección se mostrará la aplicación desarrollada con el objetivo de realizar un tratamiento de estos datos para optimizar el conocimiento de la información que aportan.

6 Sistema Web

6.1 Infraestructura

Una vez realizado el sistema en la niebla del capítulo 5, fue necesario realizar un software propio de visualización de los datos, ya que el software Z-Way está optimizado para la incorporación de dispositivos a la red ZWave y la configuración de los mismos, pero en cuanto al manejo y visualización de los datos que son aportados por estos, tiene algunas deficiencias.

Así, el sistema desarrollado tiene los siguientes objetivos generales:

- Obtención de los datos proporcionados generados por los sensores.
- Visualización y manipulación de los datos procedentes de los sensores.
- Despliegue de la información en un dominio público, accesible desde cualquier parte.

En base a los objetivos establecidos, se comenzó a diseñar un sistema capaz de cumplir los mismos de la forma más eficiente. Sin embargo, el software no se trata de un simple sistema en un equipo con unos datos proporcionados por el usuario o alguna API, sino que este software tiene que englobar varios dispositivos y fuentes de datos, por lo que se tuvo que diseñar una infraestructura (Figura 6.1) del sistema a desarrollar.

En primer lugar, los datos generados por sensores se encuentran en el módulo ZWave, conectado al dispositivo Raspberry. Para acceder a estos datos, se hace uso de una API que proporciona este módulo. Así, esta API solamente es accesible desde la misma red local en la cual está situado el dispositivo Raspberry, por lo que si se diseña un sistema web no es posible acceder a los datos de esta.

Por esa razón se hace uso de un *script* que se ejecuta contnuamente sobre Raspberry, con el objetivo de obtener los datos generados por los sensores cada vez que se generan datos en estos. Este *script* inserta los datos en una base de datos que reside en un servicio en la nube (MongoDB Atlas), por lo que los datos están accesibles desde cualquier parte.

La base de datos se trata de una base de datos MongoDB. Se ha seleccionado esta base de datos ya que se trata de una base de datos no relacional, caracterizada por ser flexible a la hora de manejar los datos y su rapidez en las consultas.

Una vez se tienen los datos en una base de datos en la nube, se realiza un sistema web encargado de acceder a estos datos mostrándolos al usuario en diferentes formas para que estos sean fácilmente entendibles y accesibles. Este sistema web se implementa haciendo uso del stack MERN. Como se ha explicado en la sección 3.6, se trata de un conjunto de tecnologías que

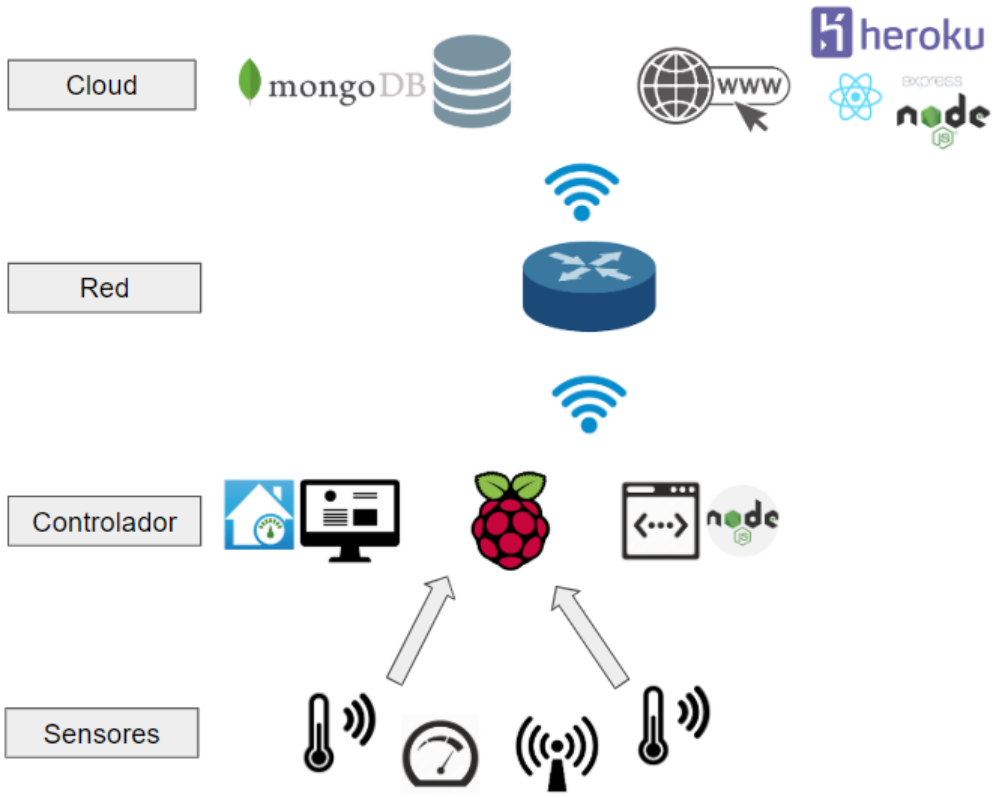


Figura 6.1: Infraestructura IoT del sistema

usan el lenguaje JavaScript, relacionadas entre sí formando una pila que permite el desarrollo de sistemas web completos usando JavaScript mayoritariamente.

En este caso, se ha decidido hacer uso del stack MERN principalmente por la base de datos que maneja, MongoDB, usada en proyectos IoT por su gran versatibilidad. Además, se hace uso de NodeJS y Express, que permiten la creación de una API en la cual se puede realizar la implementación de funciones de manejo de los datos, pudiendose hacer pública en un momento dado para desarrollar otras aplicaciones. Además, la tecnología de *front-end* se trata de React, basada en mostrar los datos que se obtiene a través de la API formada con Express y NodeJs. De esta forma, se realiza un sistema web completo que gestiona la visualización de los datos de los sensores. Esta web está desplegada en un servicio *PaaS* denominado Heroku.

La infraestructura permite la obtención de los datos del módulo ZWave, la carga de estos datos a una base de datos alojada en la nube, y la visualización de un sistema web en la que se muestran estos datos de forma clara y entendible para el usuario.

6.2 Análisis

En esta primera fase se realiza un análisis del sistema a desarrollar. En primer lugar, se realiza un análisis de la funcionalidad que debe contener el sistema web, realizando un diagrama de casos

de uso y una descripción de los casos de uso subyacentes a este. En segundo lugar, se indican los requisitos no funcionales del sistema y de qué forma se consiguen cumplir en cada caso.

6.2.1 Requisitos funcionales

La etapa de análisis del software es fundamental a la hora de desarrollar software ya que problemas en esta primera fase de análisis pueden tener consecuencias nefastas una vez que se encuentra el sistema en fase de implementación.

Con el objetivo de realizar un correcto análisis y diseño del sistema nace UML (Lenguaje unificado de modelado). UML se trata de un estándar de modelado de software, respaldado por la OMG (Object Management Group), consorcio creado para la creación y mantenimiento de estándares enfocados en tecnologías orientadas a objetos.

UML proporciona una serie de herramientas útiles para el realizar las fases de análisis, diseño y documentación del software. Concretamente, en esta sección se usan los diagramas de casos de uso.

Los diagramas de casos de uso nacen con el objetivo de recoger la interacción entre un sistema y los usuarios que pueden interactuar con este. Estos sistemas están formados por lo tanto por casos de uso, actores y relaciones entre ellos.

En la Figura 6.2 se muestra el diagrama de casos de uso de la funcionalidad incluida en el sistema.

En total, el sistema contará con la participación de dos roles de usuarios. Estos usuarios son los siguientes:

- Usuario no logueado. Se trata del usuario que no ha accedido al sistema. Para entrar en este, debe iniciar sesión. Una vez inicia sesión, deja de ser un usuario no logueado.
- Usuario. Se trata del usuario que ha iniciado correctamente sesión en el sistema y tiene acceso a la funcionalidad principal de este. Una vez cierra sesión, el usuario vuelve a ser un usuario no logueado.

A continuación se realiza una descripción de los casos de uso, mostrando los siguientes campos en cada uno:

- Nombre: Nombre del caso de uso.
- Resumen: Pequeña descripción de la funcionalidad que se lleva a cabo en el caso de uso.
- Actores principales: Actores que interactúan con el sistema en el caso de uso.
- Sistema: Sistema que implementa el caso de uso.
- Condiciones previas: Estado del usuario en el sistema previo al comienzo del caso de uso.
- Flujo de eventos: Secuencia de acciones por parte del sistema y actor en la ejecución del caso de uso.

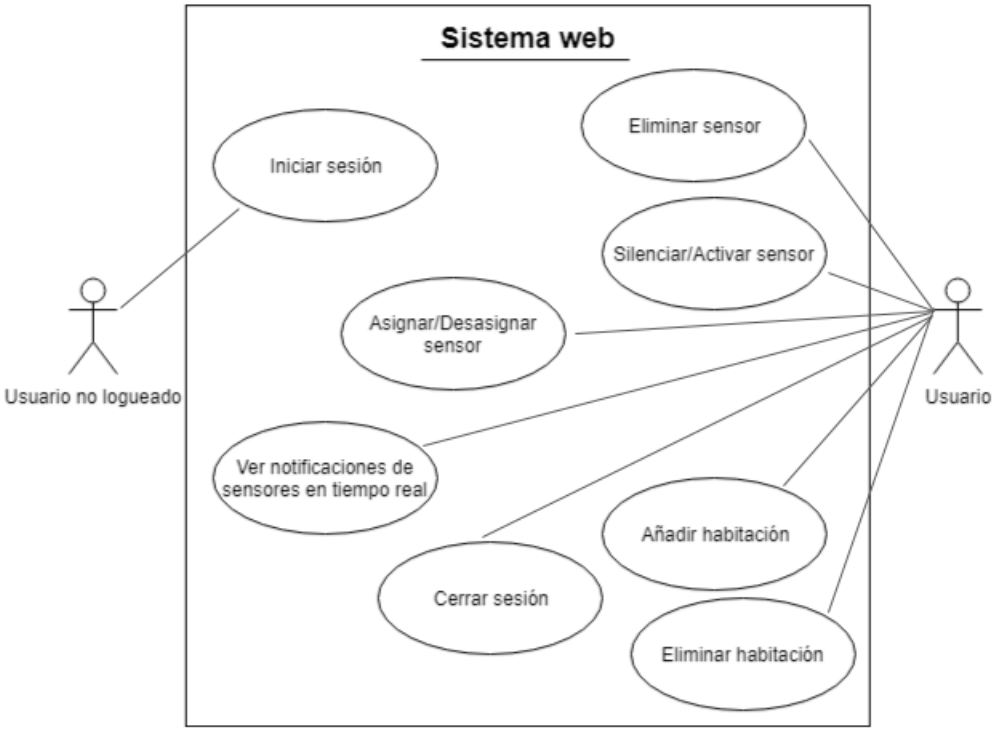


Figura 6.2: Diagrama de casos de uso del sistema web

- Condiciones de salida: Estado del sistema y actor una vez se finaliza la ejecución de la funcionalidad recogida en el caso de uso.
- Alternativas: Posibles caminos alternativos en la ejecución del caso de uso.

Propiedad	**Valor**
Nombre	Iniciar sesión
Resumen	El usuario debe iniciar sesión para poder acceder al sistema.
Actores principales	Usuario no logueado.
Sistema	Aplicación web.
Condiciones previas	El usuario se encuentra en la página de inicio de sesión en el sistema.

Propiedad	Valor
Flujo de eventos	• El sistema muestra un formulario de inicio de sesión. • El usuario introduce el email y contraseña requeridos. • El sistema valida los datos y permite al usuario entrar en el sistema.
Condiciones de salida	El sistema permite al usuario entrar en el sistema, localizándose en la página principal de este.
Alternativas	• El usuario no introduce los datos requeridos. El sistema muestra un aviso. • El usuario no introduce introduce una contraseña incorrecta o un email incorrecto. El sistema muestra un aviso. • Es la primera vez que el usuario inicia sesión. Cuando se introducen los datos, el sistema muestra un formulario en el cual se debe especificar una nueva contraseña. Si el usuario realiza la acción correcta, se vuelve al principio del caso de uso. En caso contrario, se muestra un aviso.

Tabla 6.1: Caso de uso - Iniciar sesión.

Propiedad	Valor
Nombre	Ver notificaciones de sensores en tiempo real
Resumen	El usuario podrá ver las notificaciones de los sensores ordenados por habitaciones.
Actores principales	Usuario.
Sistema	Aplicación web.
Condiciones previas	El usuario se encuentra en la ventana principal del sistema.

Propiedad	Valor
Flujo de eventos	• El sistema muestra una lista que contiene las habitaciones que hay registradas en el sistema. • El usuario hace clic en la habitación de la que quiere ver los sensores y las notificaciones que han generado estos. • El sistema muestra los sensores y notificaciones de la habitación seleccionada. • El sistema, si detecta nuevas notificaciones en las habitaciones seleccionadas por el usuario, las incorpora en la ventana en tiempo real.
Condiciones de salida	El sistema ofrece una lista de habitaciones junto con los sensores y notificaciones de la habitación seleccionada. Se indicará al usuario la habitación que está seleccionada.
Alternativas	• Si el usuario selecciona una habitación que no tiene sensores, se muestra una notificación con el mensaje "Habitación sin sensores".

Tabla 6.2: Caso de uso - Ver notificaciones en tiempo real.

Propiedad	Valor
Nombre	Añadir habitación
Resumen	El usuario podrá añadir una nueva habitación al sistema.
Actores principales	Usuario.
Sistema	Aplicación web.
Condiciones previas	El usuario se encuentra en la ventana *Gestionar habitaciones*, en la sección *Añadir habitación*.

Propiedad	Valor
Flujo de eventos	• El sistema muestra un formulario con los datos a insertar de la habitación. • El usuario introduce los datos requeridos de la habitación. • El usuario hace clic en el botón *Añadir habitación*.
Condiciones de salida	Notificación con el mensaje "Habitación añadida correctamente".
Alternativas	• El usuario no completa el campo *Nombre de habitación*, por lo tanto el sistema mostrará una notificación con el mensaje "El nombre de la habitación es obligatorio" y no se añadirá la habitación. • El usuario no acepta eliminar la actividad, por lo que no se elimina la actividad del sistema.

Tabla 6.3: Caso de uso - Añadir habitación.

Propiedad	Valor
Nombre	Eliminar habitación.
Resumen	El usuario podrá eliminar una habitación almacenada en el sistema.
Actores principales	Usuario.
Sistema	Aplicación web.
Condiciones previas	El usuario se encuentra en la ventana *Gestionar habitaciones*, en la sección *Lista*.
Flujo de eventos	• El sistema muestra un listado con las habitaciones registradas en el sistema. • El usuario selecciona una habitación para proceder a su eliminación. • El sistema muestra un mensaje de aviso de eliminación. • El usuario acepta eliminar la habitación.

Propiedad	Valor
Condiciones de salida	Notificación con el mensaje de "Habitación *nombre de habitación* eliminada correctamente". La lista que muestra las habitaciones ya no contiene la habitación eliminada.
Alternativas	• El usuario no acepta eliminar la habitación, por lo que no se elimina la actividad del sistema.

Tabla 6.4: Caso de uso - Eliminar habitación.

Propiedad	Valor
Nombre	Eliminar sensor.
Resumen	El usuario podrá eliminar un sensor del sistema para no recibir más notificaciones de este.
Actores principales	Usuario.
Sistema	Aplicación web.
Condiciones previas	El usuario se encuentra en la ventana *Gestionar sensores*.
Flujo de eventos	• El sistema muestra una lista en la cual se muestran los sensores del sistema. • El usuario selecciona el sensor que quiere eliminar del sistema y no recibir más notificaciones. • El sistema muestra una advertencia al usuario recordándole las consecuencias de la acción a realizar. • El usuario acepta definitivamente eliminar el sensor.
Condiciones de salida	Se elimina el sensor de la lista actual y no se muestra en la aplicación. No se reciben más actualizaciones de este sensor.
Alternativas	• El usuario no acepta eliminar el sensor, por lo que no se elimina la actividad del sistema.

Tabla 6.5: Caso de uso - Eliminar sensor.

Propiedad	Valor
Nombre	Asignar-Desasignar sensor
Resumen	El usuario podrá incorporar y eliminar sensores de una habitación. Además, se podrá tener sensores sin asignar a una habitación.
Actores principales	Usuario.
Sistema	Aplicación web.
Condiciones previas	El usuario se encuentra en la ventana *Gestionar habitaciones*, en la sección *Asignar-Desasignar sensores*.
Flujo de eventos	• El sistema muestra un elemento del tipo combobox en el que se encuentran todas las habitaciones del sistema. • El usuario selecciona una habitación del elemento anterior. • El sistema muestra los sensores de la habitación seleccionada. • El usuario selecciona un sensor de la lista "Sin asignar". • El sistema añade este sensor a la lista de sensores de la habitación seleccionada y elimina el sensor de la lista *Sensores sin asignar*
Condiciones de salida	Se tiene la lista de sensores de una habitación y sensores sin asignar actualizada.
Alternativas	• El usuario selecciona un sensor de la lista de "Situados". • El sistema elimina el sensor de la lista de sensores de la habitación y añade este sensor a la lista de *Sin asignar*.

Tabla 6.6: Caso de uso - Asignar-Desasignar sensor a habitación.

Propiedad	Valor
Nombre	Silenciar-Activar sensor.
Resumen	El usuario de la aplicación podrá silenciar sensores, es decir, podrá hacer que estos sensores no aparezcan en la ventana en la cual se reciben las nuevas notificaciones de los sensores. Además, esta acción se podrá deshacer para que estos sensores vuelvan a mostrarse en la ventana.

Propiedad	Valor
Actores principales	Usuario.
Sistema	Aplicación web.
Condiciones previas	El usuario se encuentra en la ventana *Sensores*.
Flujo de eventos	• El sistema muestra una lista en la cual se muestran los sensores del sistema. • El usuario desactiva el checkbox del sensor que quiere silenciar • El sistema marca el sensor como silenciado y no aparece en otras ventanas.
Condiciones de salida	Ninguna
Alternativas	• El usuario activa el checkbox del sensor que quiere volver a mostrar. • El sistema marca el sensor como activo y aparece en el resto de ventanas.

Tabla 6.7: Caso de uso - Silenciar-Activar sensor

Propiedad	Valor
Nombre	Cerrar sesión
Resumen	El usuario cierra sesión una vez quiere dejar de usar el sistema.
Actores principales	Usuario.
Sistema	Aplicación web.
Condiciones previas	El usuario se encuentra dentro del sistema web.
Flujo de eventos	• El usuario desea salir del sistema y hace click en *Cerrar sesión*. • El sistema borra los datos de la sesión actual y devuelve al usuario a la página de inicio de sesión.

Propiedad	Valor
Condiciones de salida	El sistema permite al usuario salir del el sistema, se redirige al usuario a la pantalla de inicio de sesión.
Alternativas	Ninguna

Tabla 6.8: Caso de uso - Cerrar sesión.

6.2.2 Requisitos no funcionales

En esta sección se tratan los requisitos no funcionales que tiene el sistema web desarrollado y cómo se ha conseguido cumplirlos.

Usabilidad: El sistema web tiene que ser fácil de usar e intuitivo, cualquier persona con conocimientos de informática básicos debe navegar sin problemas por ella.

Este requisito se ha conseguido insertando un menú que se muestra en todas las ventanas del sistema web. Además, se resalta con un color ligeramente diferente a la sección en la que se encuentra este. Por otra parte, se han usado iconos universales para indicar algunas acciones que se pueden realizar sobre el sistema web, por lo que las acciones posibles a realizar están indicadas.

Latencia y velocidad: El sistema web debe tener un tiempo de respuesta reducido.

Para lograr este requisito no funcional, se realiza una gran cantidad pequeñas consultas a la API para la obtención de datos, en lugar de realizar grandes consultas, incrementando el tiempo de espera en estas. De esta forma, se consigue que la información que se muestra aparecerá de forma rápida y el usuario no tendrá que esperar mucho tiempo. Un elemento que favorece esta característica es la paginación cuando se tiene una gran cantidad de datos a mostrar. De esta forma, se reduce la cantidad de datos que se muestran a la vez en la ventana, pudiendo acceder a los datos más antiguos haciendo uso de este.

Escalable y extensible: Se debe poder incluir nueva funcionalidad en el sistema web de forma sencilla.

Dado que el sistema web se ha desarrollado con el stack MERN, se pueden incluir nuevas colecciones en la base de datos MongoDB, se pueden añadir nuevas consultas en la API de forma independiente a las consultas existentes y lo más importante, se pueden añadir nuevos componentes en las ventanas del *front-end*. De esta forma, el sistema puede ser ampliado con la funcionalidad que se desee sin tener que modificar el sistema actual.

Mantenibilidad: Se deben poder realizar modificaciones en el sistema sin un alto esfuerzo.

Se ha realizado la programación del *front-end* diviendo las ventanas en componentes, de esta forma, se favorece la independencia entre ellas, favoreciendo la modificación de estas sin tener que modificar otros elementos del sistema web.

Robustez de los datos: La base de datos debe contener datos en correcto formato, sin admitir duplicados o datos inservibles.

En la base de datos se almacenan datos que no son duplicados. Así, antes de realizar una inserción o modificación de los datos se comprueba que no existe un dato igual. Esto se maneja mediante la muestra de *pop-ups*[1] al usuario en caso de error al introducir la información que proporciona al sistema por estos motivos.

Interoperabilidad: Se debe permitir el intercambio de información entre diferentes sistemas.

Dado que el sistema IoT tiene diferentes elementos, tanto en la nube como en el mundo físico, se realiza intercambio de información entre los elementos. Algunas de estas comunicaciones se basan en las APIs que se proporcionan y las que se han diseñado.

Disponibilidad: El sistema debe estar disponible en cualquier momento.

Esto se ha conseguido al desplegar el sistema web en Heroku. De esta forma, será accesible desde cualquier lugar y en cualquier momento.

Portabilidad: El sistema debe poder ejecutarse en diferentes dispositivos y sistemas.

El sistema web que se ha desarrollado se puede visualizar en dispositivos móviles, tablets y ordenadores, adaptándose su contenido para que se visualice correctamente en cada uno de estos dispositivos.

Seguridad: El sistema debe guardar la información de forma segura y permitir el acceso solamente a los usuarios con permiso para ello.

La información sensible para los usuarios como las contraseñas, se guardan haciendo uso de valores *hash*. Además, el administrador del sistema establece los usuarios con permiso para poder acceder al sistema.

6.3 Desarrollo

6.3.1 Estudio de APIs

Dado que el objetivo principal del sistema web que se ha desarrollado se trata de la visualización de los datos procedentes de los sensores, fue necesario estudiar la forma de acceder a ellos.

Como se explicó en el capítulo anterior, ha sido necesario acoplar el dispositivo Razberry para poder tener una comunicación con los sensores. Para poder administrar los dispositivos conectados mediante el protocolo ZWave, fue necesario instalar Z-Way. Este software se usó en el sistema IoT en la niebla con el objetivo de establecer la comunicación con los sensores y realizar cambios en la configuración de estos, haciendo uso de su interfaz gráfica.

Sin embargo, la principal funcionalidad del software Z-Way no reside en la interfaz gráfica que contiene, sino en la variedad de APIs diseñadas para permitir a terceros hacer uso de los datos de los sensores, con el objetivo de desarrollar sus propias aplicaciones.

[1] Ventanas emergentes o notificaciones que se muestran al usuario para avisar de que algo que ha realizado ha tenido éxito o, en caso contrario, no se puede realizar.

6.3.1.1 Estructura Z-Way

Z-Way está formado principalmente por los elementos que se muestran en la Figura 6.3. En la figura se pueden apreciar las partes de este software junto con las APIs que realizan la interacción entre estas partes Paetz (2017).

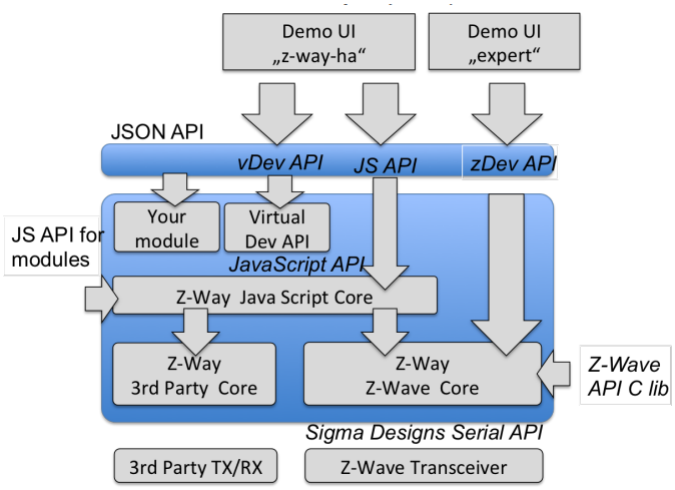


Figura 6.3: Estructura Z-Way y sus APIs

La parte más importante se trata de *Z-Way Z-Wave Core*. Se trata del núcleo de Z-Way, encargado de realizar la comunicación con el transceptor de Z-Wave. Esta comunicación se realiza a través de una API privada, *Sigma Designs Serial API*, que puede ser usada por los desarrolladores de transceptores para poder acoplar sus módulos al software Z-Way. Además, este software contiene un *servidor* de JavaScript, encargado de proporcionar una API y redireccionar las llamadas a *Z-Way Z-Wave Core*.

6.3.1.2 APIs

De esta forma, el software completo tiene 3 APIs que garantizan el acceso a los datos de los sensores que forman parte de la red ZWave administrada por este software. Las APIs son las siguientes:

Z-Way Device API: Se trata de la API que realiza una conexión directa con el núcleo del software (*Z-Way Z-Wave Core*) y sus servicios. Esta API permite la modificación de las configuraciones de la red ZWave y la realización de tareas de automatización. Hay dos versiones de esta API, con el objetivo de que pueda ser usada por un gran número de aplicaciones, dotando de ciertos permisos a los usuarios para establecer los parámetros de la red ZWave como deseen.

Z-Wave API as JSON API: Se trata de la API que se proporciona para acceder al servidor web integrado en el software, y este se comunica con el núcleo. La API consta de consultas JSON, por lo que es óptima para el lenguaje JavaScript.

Z-Wave API as C Library API: Esta API está desarrollada en *C* y por lo tanto no enlaza con el servidor de javascript integrado, sino directamente con el núcleo del software,

por lo tanto, se trata de una API un poco más óptima, ya que sus peticiones no tienen que ser redirigidas internamente.

JavaScript API: Dado que la anterior API no permite la ejecución de funciones propias para obtener datos de la red ZWave, se creó esta API. Esta, permite ejecutar funciones definidas por el usuario en el servidor de JavaScript interno.

Virtual Device API: Esta API, en lugar de definir los elementos de la red como elementos hijos de ZWave, crea un dispositivo virtual para cada elemento, permitiendo un fácil acceso a los elementos de la red.

3rd party API: Esta API implementa el mismo funcionamiento que la Z-Way Device API para otros protocolos de comunicación inalámbricos como *EnOcean*.

Una vez que se estudiaron las APIs proporcionadas, se procedió a la selección de una de ellas para realizar las consultas sobre los datos de los sensores de la red ZWave implementada.

En primer lugar, se descartó la API desarrollada en *C* ya que el lenguaje de programación en el cual se va a programar el servidor web se trata de JavaScript, por lo que no es aplicable. Así, la API JSON tampoco se ha usado, ya que no se quiere realizar ninguna configuración sobre la red ZWave.

Después, se descartó la *JavaScript API*, ya que en este caso no se quiere ejecutar funciones en el servidor web encapsulado, sino simplemente obtener los datos de los dispositivos.

Finalmente, se usó la *Virtual Device API* (*vDev API*). Con esta API, se permite acceder a las operaciones básicas de los sensores; tales como la ejecución de comandos en estos dispositivos y la extracción de los datos y notificaciones de los mismos ZWave.me (2019). Concretamente, se usa esta API de la forma que muestra el comando destacado en Código 6.1

Código 6.1: Uso vDev API. Ejecución de comando.

```
1 http://YOURIP:8083/ZAutomation/api/v1/devices/ZWayVDevice6:0:37/command/on
```

6.3.1.3 API seleccionada

De esta forma, se hace uso de *vDev API* para obtener toda la información proporcionada por el sistema IoT en la niebla que se implementó y se ha comentado en el capítulo anterior. Esta API fue desarrollada y es mantenida por *ZWave.me*, por lo que se trata de una API oficial, garantizándose el soporte de la misma.

La estructura de la API se define en ZWave.me (2019). En esta API se pueden observar consultas referentes al historial de los dispositivos, notificaciones, ubicaciones, perfiles, etc.

Una llamada a esta API puede ser la llamada *GET* que se muestra en Código 6.2.

Código 6.2: Uso vDev API. Obtención de notificaciones a partir de una fecha.

```
1 http://SERVERIP:8083/ZAutomation/api/v1/notifications?since=TIMESTAMP
```

La función principal de esta consulta se trata de devolver el conjunto de notificaciones proporcionadas por los sensores. Además, permite establecer la fecha (en formato *timestamp*) a partir de la cual se extraen estas notificaciones. La fecha se indica en el campo *TIMESTAMP* establecido en el Código 6.2.

Como resultado a esta llamada, se obtiene una respuesta un JSON con un contenido que sigue la estructura de Codigo 6.3.

Código 6.3: Respuesta zDev API

```
{
   "error": null,
   "data": {
      "updateTime": 1387884437,
      "notifications": [
         {
            "id": "1387199352223",
            "timestamp": "2013-12-16T13:09:12.223Z",
            "level": "error",
            "message": "Cannot remove location 54545 - doesn't exist",
            "redeemed": true
         },
         {
            "id": "1387200419730",
            "timestamp": "2013-12-16T13:26:59.730Z",
            "level": "error",
            "message": "Cannot remove location 54545 - doesn't exist",
            "redeemed": true
         }
      ]
   }
}
```

Como conclusión, se ha hecho uso de esta API por su facilidad de uso y su extensa documentación, más que suficiente para las acciones que se han realizado en el sistema web desarrollado. Esta elección fue fundamental, ya que el sistema web se basa principalmente en los datos proporcionados por esta API.

6.3.2 Base de datos

En el sistema IoT desarrollado, el elemento principal se trata de la base de datos. Esta base de datos se trata de una base de datos MongoDB. En esta, se almacena la información que es recogida del sistema IoT en la niebla que se ha implementado y permite la visualización en el sistema web.

La base de datos se encuentra desplegada en MongoDB Atlas. Para poder usar MongoDB Atlas, es necesario crear una cuenta gratuita. Una vez iniciado sesión, se crea una base de datos y se añaden las colecciones que va a tener esta. Como se puede observar en la Figura 6.4, MongoDB Atlas permite realizar consultas sobre los documentos de las colecciones y hacer

modificaciones en estos.

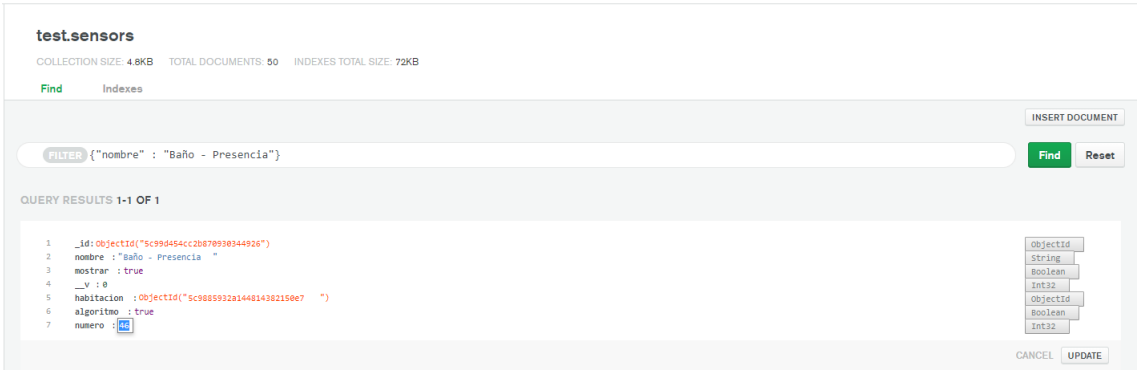


Figura 6.4: MongoDB Atlas - Consultas y modificaciones

Además, MongoDB Atlas proporciona una ventana (Figura 6.5) en la cual se pueden observar el número de conexiones que se están produciendo a esta base de datos, así como el número de operaciones que se realizan sobre los datos.

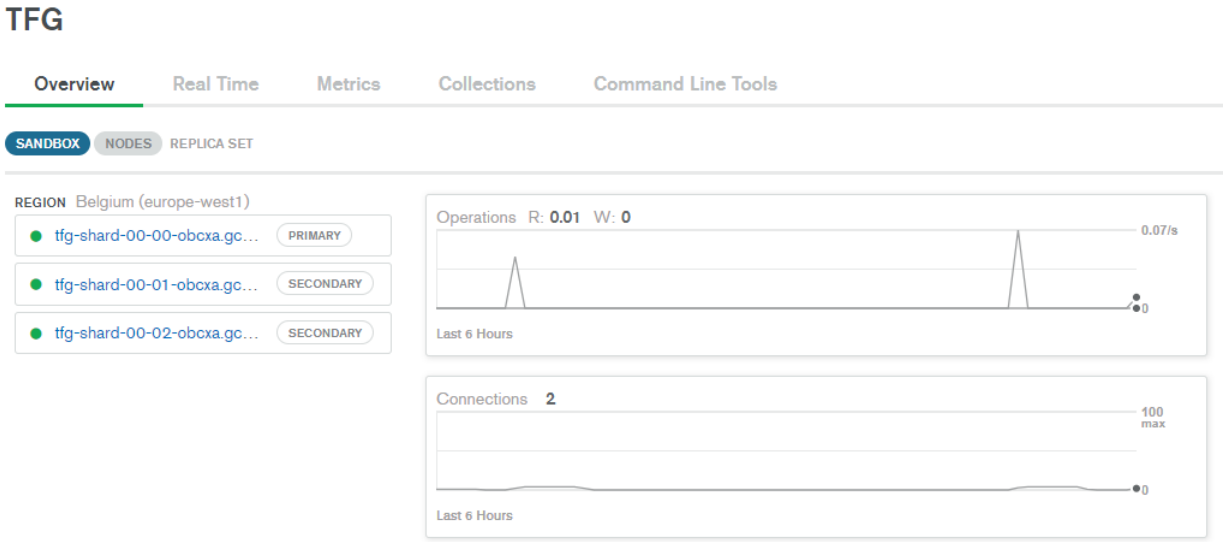


Figura 6.5: MongoDB Atlas - Conexiones

Para manejar los datos de la base de datos en el sistema web, se usa Mongoose. Mongoose se trata de un paquete de NodeJS que se instala haciendo uso de *npm*, y consiste en una librería que permite la creación de esquemas para una base de datos MongoDB. Estos esquemas hacen que la base de datos se pueda manejar como una base de datos relacional. Además, proporciona un *ORM*[2] bastante útil a la hora de realizar consultas sobre los datos, por lo que no se tiene que usar la sintaxis de MongoDB, que es algo más complicada.

[2]Mapeo objeto-relacional. Permite la definición de esquemas sobre una base de datos con el objetivo de acceder a los datos de esta de forma consistente

La información que se almacena en la base de datos MongoDB se trata de información normalizada, es decir, se definen esquemas con Mongoose y los registros que se inserten tienen que cumplir con las restricciones que se establecen en él. Así, si se desea incorporar un documento a la colección *Ejemplo*, se deberán respetar las restricciones del esquema que se puede visualizar en el Código 6.4.

Código 6.4: Esquema de ejemplo en Mongoose

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;
var ejemploSchema = new Schema({
    ejemplo: {type: String, required: true, max: 100},
});
module.exports = mongoose.model('Ejemplo', EjemploSchema);
```

En este instante, puede plantearse la duda de por qué se ha usado en realidad MongoDB en este sistema web. Se decidió hacer uso del stack MERN para realizar la programación del sistema; este stack establece que la base de datos se trata de MongoDB. Dado que se usa Mongoose, se trata como una base de datos relacional, por lo que no hay diferencia entre usar esta base de datos u otra.

En MongoDB, la información que se almacena en la base de datos se realiza en forma de documentos agrupados en colecciones. Las colecciones que son usadas por el sistema web son las siguientes:

Notificación En esta colección se almacena la información referente a las notificaciones que generan los sensores desplegados en la vivienda. Cada vez que se produce un cambio en la propiedad que mide un sensor, se envía una notificación al módulo Razberry y es almacenada en este. Los campos principales que contiene cada documento de esta colección se han definido como se muestra en el Código 6.5

Código 6.5: Esquema Notificacion

```
var NotificacionSchema = new Schema({
    valor: {type: String, required: true, max: 100},
    fecha_creada_parseada: {type: String, required: true},
    sensor: {type: Schema.Types.ObjectId, ref: 'Sensor', required: false},
    valor_actualizacion: {type: Number, required: true},
});
```

Como anotación, el campo *fecha_creada_parseada* almacena la fecha de creación de la notificación en formato correcto para su visualización (horas:minutos:segundos dia/mes/año) en el sistema web. Se almacena este registro en la base de datos ya que si no se hace, cada vez que se obtienen las notificaciones en el sistema web, se debería realizar la transformación del valor almacenado en *valor_actualizacion* (*timestamp* en formato numérico) en una cadena de texto con la fecha entendible, gastando bastante tiempo y haciendo trabajo repetitivo cada vez que se obtienen las notificaciones.

Habitacion En esta colección se almacenan las habitaciones que define el usuario para ubicar los distintos sensores (Código 6.6).

Código 6.6: Esquema Habitacion

```
var Habitacion = new Schema({
    nombre_habitacion: {type: String, required: true},
    fecha_creada: {type: Date, required: false},
});
```

Sensor En esta colección se almacenan los sensores que se han conectado al módulo Razberry alguna vez y contienen alguna notificación (Código 6.7).

Código 6.7: Esquema Sensor

```
var SensorSchema = new Schema({
    nombre: {type: String, required: true, max: 100, unique:true},
    habitacion: {type: Schema.Types.ObjectId, ref: 'Habitacion', required: false},
    mostrar : {type: Boolean, required: false},
});
```

Usuario En esta colección se almacenan los usuarios que pueden acceder al sistema web (Código 6.8).

Código 6.8: Esquema Usuario

```
var UsuarioSchema = new Schema({
    email: {type: String, required: true, max: 100, unique:true},
    pass: {type: String, required: true, max: 100, unique:true},
    validado : {type: Boolean, required: false},
});
```

El campo *validado* se usa para saber si el usuario ha accedido por primera vez y ha cambiado la contraseña de forma satisfactoria. El campo pass almacena un valor *hash* que codifica la contraseña del usuario haciendo uso de *HMAC SHA-256* (Código 6.6).

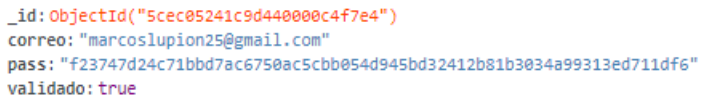


Figura 6.6: Documento usuario

6.3.3 Funcionamiento stack MERN

Como ya se ha mencionado, para realizar el sistema web se ha usado el stack MERN. A continuación se explica qué función tiene cada tecnología en el sistema web que se ha desarrollado.

React se basa en la realización del *front-end* usando componentes. Estos componentes pueden entenderse como un equivalente a las clases en un lenguaje orientado a objetos, ya que constan de atributos, funciones y otros elementos propios de la orientación a objetos. Estos componentes contienen una función principal denominada *render()*, en la cual se devuelve código HTML que se muestra en la web. Sin embargo, este código HTML se genera usando JSX, un lenguaje que permite la incorporación en el código HTML de datos dinámicos procedentes de llamadas a alguna API. Así, los componentes de React se tratan de entidades que interactúan con una API y muestran los datos que se obtengan de esta. Así, estos podrán procesar los datos obtenidos e implementar funciones para lograr una mejor interacción con el usuario.

En cuanto a la visualización de los elementos HTML, se ha hecho uso del *framework* Materialize, que proporciona clases CSS aplicables a los elementos HTML con el objetivo que la interfaz tenga un diseño similar al usado por Google en sus aplicaciones Prabhu y Shenoy (2016). De esta forma se favorece la usabilidad de las nuevas aplicaciones, al tener el mismo estilo que Google y por lo tanto, ser usadas por una gran cantidad de personas.

ExpressJS y NodeJS proporcionan las funciones para crear una API que es usada por el *front-end* para mostrar los datos procedentes de la base de datos. Esta API contiene todas las funciones necesarias por los componentes React para realizar una correcta visualización de los datos. Para cada nueva consulta a los datos es necesario realizar una función en la API. Las funciones pueden ser de 4 tipos, *POST*, *GET*, *PUT* y *DELETE*. Dependiendo de si la función es insertar, obtener, modificar o eliminar un registro, se usará un tipo de función u otro.

MongoDB se trata de la base de datos en la cual se almacenan los datos de los sensores y los datos que se manejan en el sistema web, como pueden ser las habitaciones que define el usuario para ubicar los sensores. A esta base de datos se accede a través de la API desarrollada en ExpressJS y NodeJS, por lo que no se puede acceder directamente desde el *front-end*.

6.3.4 Script obtención nuevas notificaciones

Una vez estudiada la API a usar para obtener los datos proporcionados por los sensores, se programó un *script* cuya función principal es obtener los cambios que se producen en los sensores e insertarlos en la base de datos. Por lo tanto, como uno de los objetivos del sistema es estar disponible las 24 horas del día, este *script* tiene que estar ejecutándose siempre.

Dado que el software Z-Way se ejecuta en el dispositivo Razberry colocado en los puertos *GPIO* de la Raspberry, este *script* se ejecuta en la Raspberry.

La programación del *script* se hace en el lenguaje de programación JavaScript; concretamente, se programa un *script* NodeJS, ya que la API devuelve objetos JSON y este lenguaje permite una buena manipulación de objetos de con este formato. Los pasos detallados que se realizan en este *script* son los siguientes:

Conexión con la API En primer lugar, se realiza una llamada a la API, concretamente la llamada que se muestra en el Código 6.9. Se trata de una llamada *POST*, en la cual se envía el nombre de usuario y la contraseña que se ha establecido en el software Z-Way. Se realiza esto para obtener una *cookie*[3], que se envía con cada consulta para no tener que iniciar sesión cada vez que se quiera realizar una nueva consulta a la API.

En cuanto a la URL, se especifica en el parámetro *dirección* el valor de *localhost*, ya que Z-Way se está ejecutando en la misma máquina donde se está ejecutando el *script*, concretamente en el puerto 8083. Una vez se hace esta llamada, el servidor de Z-Way responde con un estado de 200 si el *inicio de sesión* es correcto, devolviendo la *cookie*.

Código 6.9: Login en la API

```
var options = {
   uri: 'http://' + direccion + ':8083/ZAutomation/api/v1/login',
   method: 'POST',
   json: true,
   body: {
   login: 'admin',
   password: 'contrasenia'
}
```

Una vez realizado el inicio de sesión, se almacena la variable de sesión o *cookie* en una variable accesible en todo el *script*, y se enviará con cada nueva consulta.

Conexión a la base de datos Antes de realizar la primera consulta para obtener los datos de las notificaciones de los sensores, se realiza la conexión con la base de datos MongoDB desplegada en MongoDB Atlas. La conexión a la base de datos se realiza de la forma que se puede observar en el Código 6.10.

En primer lugar, dado que la base de datos se encuentra alojada en el servicio MongoDB Atlas, se ha creado la cadena que realiza la conexión con esta base de datos, especificando el nombre de usuario y la contraseña.

Dado que se usa la librería Mongoose para manejar el contenido de la base de datos, una vez se realiza la conexión con Mongoose, esta conexión perdura en toda la ejecución del *script*, ya que se permiten hasta 100 conexiones simultáneas en MongoDB Atlas, y en este sistema, nunca se llega a ese número de conexiones.

Código 6.10: Conexión a la base de datos

```
const uri = "mongodb+srv://marcos:PASSWORDG@tfg-obcxa.gcp.mongodb.net/test?↩
    ↪ retryWrites=true";
const mongoose = require('mongoose');
mongoose.connect(uri , {useNewUrlParser : true , useFindAndModify:false,↩
    ↪ useCreateIndex : true})
```

[3]Una cookie se trata de un conjunto de información que se intercambia entre el servidor web y el usuario para mejorar la experiencia del usuario en la web, permitiéndole al servidor guardar información de este

```
    .then(db => console.log('DB Connected'))
    .catch(err => console.error(err));
module.exports = mongoose;
```

Obtención de nuevas notificaciones Una vez que se ha realizado la conexión con la base de datos y con el servidor Z-Way, se puede realizar la principal función del *script*, insertar las nuevas notificaciones en la base de datos. Sin embargo, en la API no existe ninguna consulta que permita obtener las notificaciones a partir de la última notificación que se ha devuelto en una consulta previa, por lo tanto, el *script* tiene que saber en todo momento cuál es la notificación más reciente que se tiene, con el objetivo de obtener las notificaciones posteriores a esta.

Así, antes de realizar la llamada a la API, se tiene que obtener la notificación más reciente almacenada en la base de datos. Una vez que se tiene esta notificación, se realiza la llamada a la API que se muestra en el Código 6.11.

Código 6.11: Consulta de notificaciones a la API

```
var options2 = {
    url: 'http://' + direccion + ':8083/ZAutomation/api/v1/notifications?since=' +↩
        ↪ timestamp_reciente,
    method: 'GET',
    dataType: 'json',
    headers: {
    'Content-Type': 'application/json',
    Cookie: cookie,
};
```

Esta llamada se trata de una solicitud de tipo *GET*, en la cual se envía la *cookie* obtenida en el Código 6.9. Además, se especifica el valor de *timestamp* de la notificacion más reciente almacenada en la base de datos para obtener solamente las notificaciones más recientes a esta.

Como resultado a esa consulta HTTP, se obtiene una respuesta (en el caso de haber notificaciones más recientes) como se puede observar en el Código 6.12.

Código 6.12: Respuesta Notificaciones

```
{
"data": {
    "updateTime": 1557157779,
    "notifications": [
        {
            "id": 1557075066034,
            "timestamp": "2019-05-05T16:51:06.034Z",
            "level": "device-info",
            "message": {
            "dev": "Baño - Presencia",
            "l": "22.8 °C",
            "location": ""
        },
        "type": "device-temperature",
```

```
        "source": "ZWayVDev_zway_2-0-49-1",
        "redeemed": false
        },
        {
            "id": 1557075066161,
            "timestamp": "2019-05-05T16:51:06.161Z",
            "level": "device-info",
            "message": {
            "dev": "Baño - Ducha - Luminosidad",
            "l": "0 Lux",
            "location": ""
        },
        "type": "device-luminiscence",
        "source": "ZWayVDev_zway_2-0-49-3",
        "redeemed": false
        }
    ]
}
```

Procesamiento de la respuesta Tal y como se puede apreciar en el Código 6.12, las respuestas de la API vienen dadas en un objeto cuyo formato es JSON. Para insertar estos objetos en la base de datos MongoDB, se tiene que realizar un procesamiento de la información que viene dada en el objeto JSON.

Una vez obtenidos los datos del objeto JSON y se realizan las modificaciones oportunas (por ejemplo, convertir a mayúscula el valor que contiene el estado del sensor), se realiza la inserción de la notificación en las tablas de la base de datos Notificacion.

Ejecución infinita Dado que el *script* tiene que estar constantemente obteniendo las nuevas notificaciones, se incorpora un bucle infinito en él. Concretamente, se hace uso de la función que se muestra en el Código 6.13. Esta función incluye las acciones que se han descrito a partir del apartado 6.3.4, ya que los dos primeros pasos se realizan solamente la primera vez que se ejecuta el *script*. Así, la función de este código consiste en ejecutar el contenido dentro de *function* cada X milisegundos. En este caso, se han establecido 1000 milisegundos.

Código 6.13: Función en bucle

```
setInterval(X, function(params){
    //contenido
})
```

Planificación con la utilidad *cron* El *script* que se ha diseñado se ejecuta siempre que la Raspberry se encuentra encendida. Para hacer que esto sea así, se programó una tarea usando la utilidad *cron* de Linux. Con esta utilidad se puede programar la ejecución de *scripts* de forma automática cuando el usuario desee. Así, la configuración de la tabla de *cron* se ha completado de

la forma que se puede observar en el Código 6.14. Concretamente, se ha programado la ejecución del *script* cada vez que se reinicia el dispositivo. De esta forma, siempre que el dispostivo está encendido, se ejecuta el *script*.

Código 6.14: Tarea en cron

```
@reboot node /home/pi/Desktop/ClassifyActions/ClassifyActionsScript/src/index.js
```

6.3.5 API desarrollada en Express y NodeJS

El sistema web desarrollado ha seguido el stack MERN. En este, es necesario que las ventanas que React ofrece tengan los datos correctos en cada caso. Así, la comunicación entre React y la base de datos MongoDB se realiza a través de una API que se tuvo que implementar. Concretamente, para realizar la API se usa Express y NodeJS.

Así, la API consta de un conjunto de *endpoints*[4] accesibles por el *front-end*, encargados de realizar las consultas a la base de datos. Como ya se ha explicado, estos *endpoints* pueden ser de 4 tipos, dependiendo del tipo de consulta que se realice en cada caso.

En cuanto al contenido de las funciones que se encuentran en la API, estas funciones se encargan de realizar la interacción con la base de datos. En este caso, se usa el paquete Mongoose, ya que proporciona un ORM bastante útil a la hora de realizar las consultas a esta. Además, estas funciones pueden aceptar parámetros, lo que permite personalizar las consultas dependiendo de los valores que se obtengan a través de estos.

GET En el código 6.15 se puede observar una consulta a la API desarrollada. En este caso se trata de una consulta *GET*, ya que se realiza un listado de información procedente de la base de datos, sin hacer modificaciones en esta. Así, para poder usar los elementos de una colección con Mongoose, se tiene que acceder al esquema que se define; en este caso, se quiere hacer una consulta sobre los registros de la colección Habitacion. El ORM de Mongoose proporciona funciones como *.find()*, encargadas de obtener los documentos que cumplan ciertos requisitos.

Código 6.15: Endpoint GET en la API

```
router.get('/obtener_habitaciones_totales', async (req, res) => {
    const Habitacion = require('../models/Habitacion.js');
    var registros = await Habitacion.find();
    var devolver = {};
    console.log("Entra en el bucle")
    for (r in registros) {
        var datos = registros[r];
        devolver[datos._id] = datos.nombre_habitacion;
    }
    res.json({ devolver });
});
```

[4]Se trata de la función o parte del servidor en la que se manejan las peticiones a la API.

En este caso, se obtienen todos los documentos que contiene esta colección. Otra característica de las funciones de la API se trata de la respuesta que se proporciona. Como se puede observar, la respuesta se especifica en un objeto con formato JSON. En este caso, el objeto JSON es el que se muestra en el Código 6.16.

Código 6.16: Objeto JSON con respuesta de la API

```
{
    "devolver": {
        "5c9885932a144814382150e7": "Cuarto de baño",
        "5ca6f246a1012d17485e8553": "Salon",
        "5ca6f24ca1012d17485e8554": "Cocina",
        "5caae41d769dba291c6490f6": "Sin asignar",
        "5cb43b7966f03c1158492d80": "Exterior",
        "5cb43b7c66f03c1158492d81": "Dormitorio",
    }
}
```

POST A continuación, en el Código 6.17, se puede observar una consulta *POST* de la API desarrollada. Este tipo de consulta se realiza para realizar inserciones en la base de datos de información que procede del usuario o de la aplicación que realiza la llamada a esta función de la API. Así, junto la llamada a esta función se realiza el envío del objeto JSON con los datos a insertar (Código 6.18).

Código 6.17: Endpoint POST en la API

```
router.post('/insertar_habitacion', async (req, res) => {
    const nombre = req.body.habitacion;
    const data2 = { nombre_habitacion: nombre };
    var acc = await Habitacion.find(data2);
    if (acc.length > 0) {
        res.json({ status: 'No se puede insertar una accion existente' });
        return;
    }
var habitacion = new Habitacion(data2);
habitacion.fecha_creada = new Date();
await habitacion.save();
res.json({ status: 'Habitacion guardada' });
});
```

Código 6.18: JSON de datos para la consulta POST

```
{
    "habitacion" : "Garaje"
}
```

En la función de la API se accede al contenido de este objeto haciendo uso de la propiedad *req.body*, que contiene los atributos de este objeto JSON. Así, se realiza una consulta en la cual se obtienen las habitaciones con el mismo nombre que se obtiene del objeto que se recibe. En el caso

de que exista, se devuelve un mensaje al usuario notificando del error que se ha producido. En caso contrario, se realiza una inserción en la base de datos. Esta inserción se realiza creando un objeto especificando los atributos de este y ejecutando finalmente la función *save()*, que inserta en elemento en la base de datos.

DELETE En el código 6.19 se puede observar un caso en el que la función de la API es del tipo *DELETE*. En este caso, se obtiene el *id* de la habitación que se quiere eliminar usando el parámetro *:id* que se especifica en las propiedades de la función de la API, concretamente *req.params*. A continuación, se realiza la eliminación de la habitación haciendo uso de la función *findByIdAndDelete(id)*. Finalmente se informa al usuario del éxito de esta eliminación.

Código 6.19: Endpoint DELETE en la API

```
router.delete('/eliminar_habitacion/:id', async (req, res) => {
    var id = req.params.id;
    var conditions = { habitacion: id }, update = { habitacion: null }, options = ↩
        ↪ { multi: true };
    await Sensor.update(conditions, update, options);
    await Habitacion.findByIdAndDelete(id);
    res.json({ resultado: "correcto" })
})
```

Como se ha podido observar, la realización de la API incluye una gran variedad de consultas a las colecciones de la base de datos. Sin embargo, en este sistema web no se ha creado ninguna función en la API de tipo *PUT* (modificación).

Funciones API Dado que el objetivo de esta memoria no consiste en explicar profundamente lo que hace cada función de la API, a continuación se muestran las direcciones de las funciones de la API junto con el tipo de consulta que se trata y una pequeña explicación del objetivo que tiene cada función (Tabla 6.9). Las consultas se muestran en orden alfabético.

Tipo	Ruta	Descripción
GET	/actualizar_habitacion/:hab/:canal	Se monitorizan las notificaciones de una habitación estableciendo el canal en el cual se insertan las nuevas notificaciones.
DELETE	/eliminar_habitacion/:id	Se elimina una habitación del sistema. Los sensores situados en esta estarán sin asignar a partir de este momento.
GET	/eliminar_sensor_habitacion/:sensor	Se desasigna el sensor especificado de la habitación en la que se encuentra.
POST	/establecer_contrasenia	Se establece una nueva contraseña para el usuario una vez inicia sesión por primera vez.

Tipo	Ruta	Descripción
GET	/habitaciones	Se obtienen las habitaciones del sistema ordenadas por antiguedad creciente.
GET	/habitaciones/:saltadas	Se devuelven las seis siguientes habitaciones a partir de la habitación especificada. En primer lugar se obtienen las habitaciones más recientes.
POST	/iniciar_sesion	Se comprueban las credenciales del usuario para poder iniciar sesión.
POST	/insertar_habitacion	Se añade una nueva habitación en el sistema.
POST	/insertar_sensor_habitacion	Se asigna una habitación a un sensor.
GET	/notificaciones/:sensor/:saltadas	Se devuelven las notificaciones de un sensor
GET	/obtener_habitaciones_totales	Se obtienen todas las habitaciones de la vivienda.
GET	/sensor/gestionar_mostrar/:id_sensor	Se cambia el estado del sensor. Si está mostrándose deja de mostrarse y si no se muestra, se habilita para que se muestre.
GET	/ultimas_notificaciones/:habitacion	Se obtienen las últimas notificaciones de una habitación dada.

Tabla 6.9: Lista de endpoints de la API.

6.3.6 Sistema web

En esta sección se muestra el sistema web desarrollado junto con una explicación de los aspectos más relevantes de este.

6.3.6.1 Estructura general

La estructura general en la que se ha distribuido el sistema web es la siguiente:

- Visualización de los datos por habitaciones
  - Habitación 1
    - Últimas notificaciones
    - Sensor 1
      - Notificación 1
      - Notificación 2
      - Notificación 3

        * Sensor 2
        * Sensor 3
    - Habitación 2
    - Habitación 3
- Gestión de habitaciones
    - Añadir habitación
    - Asignar sensor en habitación
    - Listar habitaciones
    - Eliminar habitación
- Gestión de sensores
    - Listar sensores
    - Listar sensores silenciados

6.3.6.2 Iniciar sesión

El sistema web es accesible para todos los usuarios de Internet, ya que se encuentra desplegado en un dominio público. Dado que es imprescindible proteger la información que se genera en la vivienda, se establece una ventana en la cual inician sesión los usuarios dados de alta por el administrador del sistema (Figura 6.7).

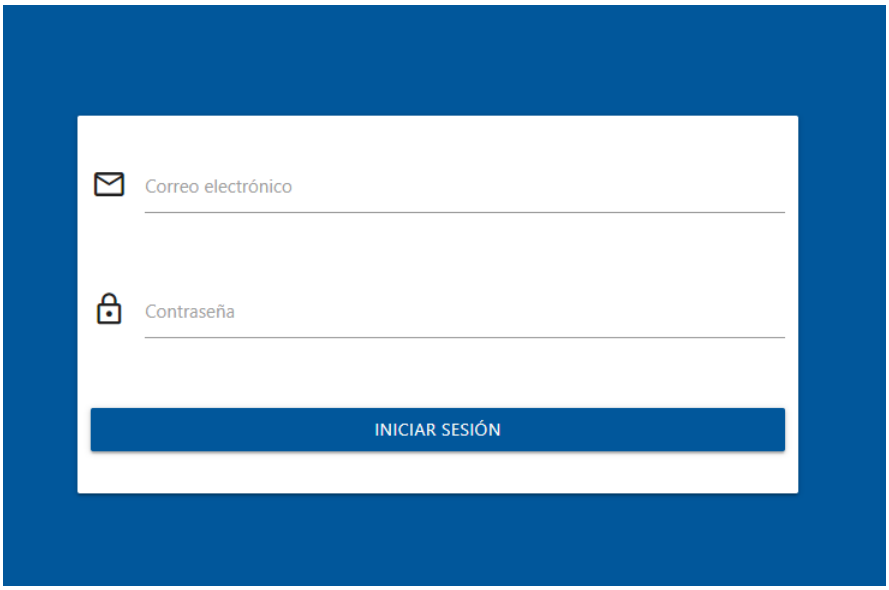


Figura 6.7: Inicio de sesión

Para dar de alta un usuario, se hace directamente desde MongoDB Atlas. Concretamente, se crea un documento de la colección *Usuarios* en el cual se establece el correo electrónico de la

persona que puede acceder al sistema junto con una contraseña que establece el administrador (Figura 6.8).


```
_id: ObjectId("5cec05241c9d440000c4f7e4")
correo: "marcoslupion25@gmail.com"
pass: "f23747d24c71bbd7ac6750ac5cbb054d945bd32412b81b3034a99313ed711df6"
validado: true
```


Figura 6.8: Usuario almacenado en la base de datos

La primera vez que el usuario accede al sistema, se requiere el cambio de la contraseña para continuar de forma segura (Figura 6.9).

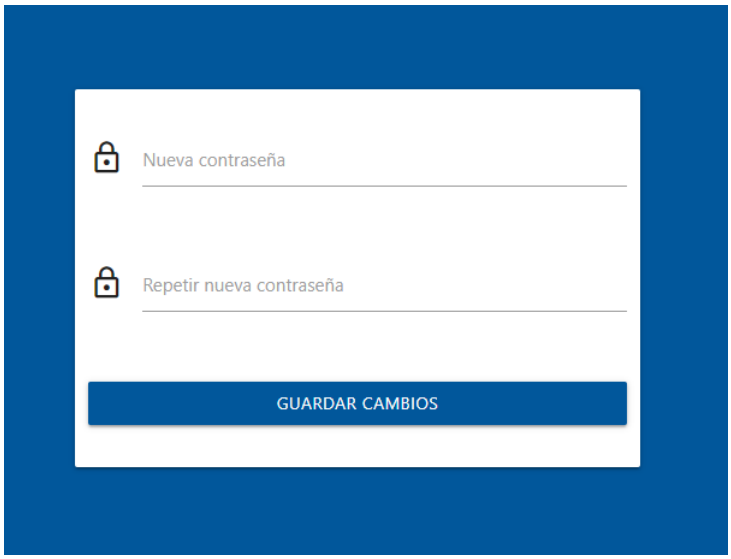


Figura 6.9: Cambio de contraseña

La contraseña que establece el usuario se guarda la base de datos haciendo uso de un valor *hash* usando HMAC SHA-256. Para generar hash, se hace uso de la librería Crypto. Cuando el usuario introduce la contraseña en el formulario de inicio de sesión, el sistema obtiene el valor hash de esta haciendo uso de la función *createHmac(valor,"codificacion_secreta")* y comprueba que el valor *hash* se corresponda con el campo *pass* del documento con el *email* especificado en el formulario (Código 6.20). De esta forma, no se guardan las contraseñas en la base de datos, sino valores *hash*, más seguros.

Código 6.20: Comprobación de contraseña

```
1 var CryptoJS = require("crypto-js");
2 var contrasenia = CryptoJS.HmacSHA256(pass, "clave").toString(CryptoJS.enc.Hex)
3 if (contrasenia == Usuario.find({email : usuario})[0].pass){
```

```
    //inicia sesión
}else{
    //no inicia sesion
}
```

Para gestionar el inicio de sesión y el mantenimiento de esta, se hace uso del paquete *express-session*. Este paquete permite crear variables de sesión. De esta forma, se permite al usuario que ha iniciado sesión tener acceso a las distintas ventanas a diferencia de los usuarios que no lo han hecho.

Así, antes de mostrar cada ventana, se hace una consulta a la API en la cual se comprueba que el usuario ha iniciado sesión. En cada *endpoint* se comprueba que el usuario ha iniciado sesión haciendo uso de la función definida en el Código 6.21.

Código 6.21: Comprobación logueado

```
function logueado(){
    return (req.session.email != undefined)
}
```

6.3.6.3 Notificaciones de sensores

En esta ventana se realiza la funcionalidad principal del sistema, obtener los datos en tiempo real de los sensores de la vivienda. En esta, se muestran las diferentes habitaciones (Figura 6.10) que componen la vivienda en la cual se ha implementado el sistema. Así, estas habitaciones en un principio se muestran en forma de lista, de esta forma el usuario puede seleccionar la habitación de la cual quiere ver las últimas actualizaciones de los sensores. No se muestra el contenido completo de todas las habitaciones porque el tiempo de espera del usuario tendría que ser elevado para ver todo el contenido a la vez.

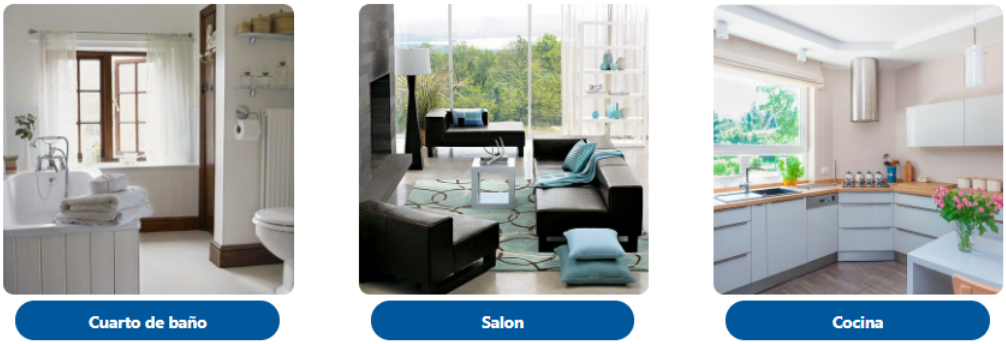


Figura 6.10: Lista de habitaciones

Así, el componente principal de esta ventana se encarga de obtener el conjunto de habitaciones y crear un componente *hijo* para cada habitación (Código 6.22). De esta forma, se tienen tantos componentes *hijos* como habitaciones tiene el sistema registradas.

Código 6.22: Creación de componentes hijos

```
render() {
   return (
      <div className="contenedor">
         <div className="row row_sin_margin_debajo col s12" >
            {Object.keys(this.state.habitaciones).map(clave => (
               <div className="col s12">
                  <Habitacion nombre={this.state.habitaciones[clave]} action={↩
                     ↪ this.childHandler} />
               </div>
            ))}
         </div>
      </div>
   )
}
```

Así, se tiene un conjunto de habitaciones encargadas de mostrar los datos procedentes de los sensores de cada habitación. Estos componentes hijos están constantemente buscando actualizaciones en la base de datos correspondientes a los sensores que abarca cada habitación. Cuando el usuario se encuentra en esta ventana, puede acceder a los datos de las habitaciones pulsando sobre ellas. Así, cuando se abre una habitación, las actualizaciones que ocurren en esta se muestran en tiempo real con un color diferente a las notificaciones antiguas. De esta forma, el usuario no tiene que recargar la página cuando se generan nuevas notificaciones.

Monitorización en tiempo real Para realizar esta monitorización constante de nuevas notificaciones se hace uso de Pusher, que permite la creación de canales de comunicación constante entre sistemas de *front-end* y APIs situadas en el *back-end*. Además, se realiza una conexión a la base de datos permanente, por lo que en cuanto el dispositivo Raspberry inserte un documento en la colección de notificaciones, se obtiene de inmediato el documento gracias a esta conexión realizada en el *back-end*. El objetivo de cada componente de habitación consiste en realizar esta conexión con la base de datos permanente para obtener las notificaciones que pertenecen a dicha habitación y mostrarlas en tiempo real en el sistema web.

Para realizar el sistema en tiempo real ha sido necesario llevar a cabo las siguientes acciones:

- Creación del objeto Pusher. Una vez se crea el componente de la habitación, en la API se realiza la creación del objeto Pusher (Código 6.23).

Código 6.23: Creación objeto Pusher

```
var Pusher = require('pusher');
var pusher = new Pusher({
   appId: 'ID',
   key: 'KEY',
   secret: 'PASS',
   cluster: 'eu',
   encrypted: true
});
```

- Realización de la conexión de monitorización con MongoDB. Se ha hecho uso de la función *watch()*, para establecer la conexión con la colección seleccionada (notificaciones en este caso). Además, se establecen los parámetros de los eventos que se reciben para realizar un filtrado sobre estos y solamente obtenga las notificaciones de la habitación en cuestión. Así, las restricciones (Código 6.24) son las siguientes:
  - Tipo de operaciones *insert*. Solamente se desea obtener los documentos que se inserten.
  - Habitación de las notificaciones. Dado que cada habitación realiza una conexión permanente con la base de datos, en cada habitación se obtienen solamente las notificaciones que se corresponden con sensores ubicados en esta.

Código 6.24: Restricciones función watch()

```
pipeline = [{
   $match: { sensor: { $in: sensores_habitacion } },
   $match: {"operationType" : "insert" }
}];
changeStream = collection.watch(pipeline);
```

- Creación del canal e inserción de notificaciones en este. Una vez se recibe una nueva notificación de la base de datos que supera los filtros, se inserta en el canal de Pusher (Código 6.25) que se crea para esta habitación, estableciendo como nombre del canal el nombre de la habitación que realiza la consulta a la API. De esta forma, una vez se inserta la notificación en el canal, esta notificación llega al *front-end*.

Código 6.25: Inserción de notificación en el canal

```
changeStream.on("change", function (change) {
   pusher.trigger(
       'notificaciones',
       req.params.id_canal,
       change.fullDocument
   );
});
```

- Manejo de la nueva notificación en el *front-end*. Una vez la notificación llega al *front-end*, se actualizan los datos del sistema web con la nueva notificación. De esta forma, cada vez que se produce una nueva notificación no se tiene que realizar una consulta a la API que traiga todos los datos de las notificaciones de la habitación. Gracias a esta funcionalidad, se reduce el tráfico entre el *front-end* y la API, reduciendo los tiempos de espera para el usuario y favoreciendo un mayor dinamismo en el sistema web.
- Cierre del canal. Cuando se desea finalizar el *streaming* de datos, desde el *front-end* se emite una señal a ejecución de la consulta a la API (Código 6.26) indicando que esta conexión debe finalizar (Código 6.27). Una vez la API recibe este aviso, ejecuta el código asociado a este evento (Código 6.28). En él, se realiza la desconexión de la API del canal Pusher y también se finaliza la conexión en streaming con la base de datos.

Código 6.26: Llamada a la API

```
fetch('/actualizar_datos_habitacion/' + this.state.nombre_habitacion ,{ signal : this.controller.signal })
    .then(res => res.json())
```

Código 6.27: Aviso finalización streaming

```
this.pusher.disconnect();
this.controller.abort();
```

Código 6.28: Detención de la ejecución de watch()

```
req.on('close', async () => {
    changeStream.close();
    pusher.disconnect();
    res.json({status : "Cerrado"})
});
```

Por otra parte, la habitación una vez se encuentra desplegada, se divide en dos secciones, que se muestran a continuación.

Últimas actualizaciones En esta sección se muestra una lista con las últimas 10 actualizaciones (Figura 6.11) que se han producido en la habitación abierta. Las actualizaciones que se muestran en esta sección proceden de todos los sensores de la habitación. Esta funcionalidad se ha implementado ya que en algunas ocasiones es necesario visualizar las notificaciones que ocurren en una habitación de todos los sensores a la vez, para poder detectar ciertos patrones entre sensores y no tener los datos de los sensores aislados entre sí como el software Z-Way.

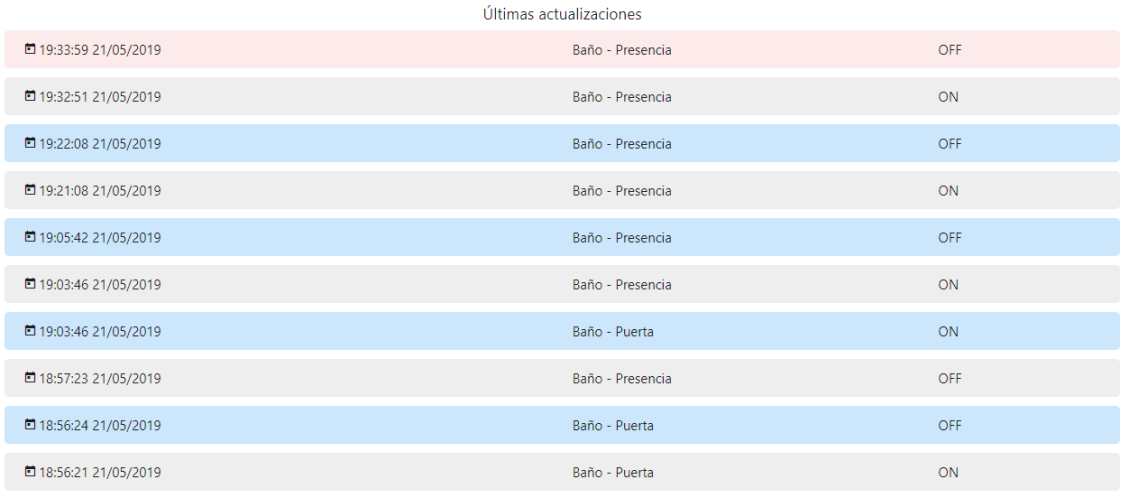


Figura 6.11: Últimas actualizaciones de la habitación

Sensores Cuando se abre la habitación también se observa una lista que contiene los sensores que están ubicados en la misma (Figura 6.12).

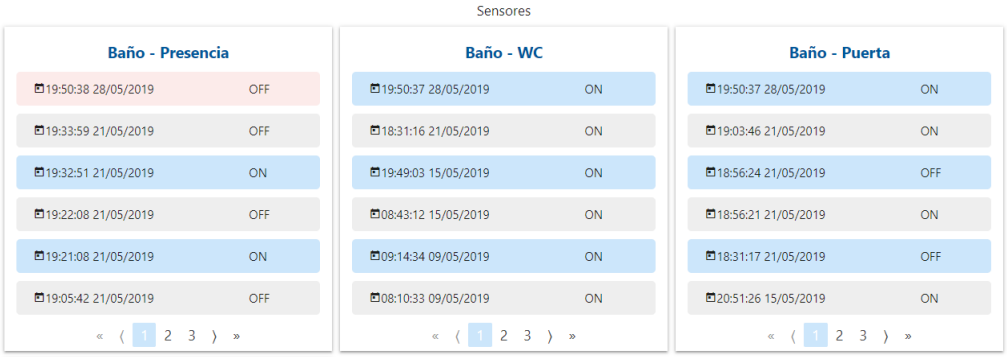


Figura 6.12: Sensores ubicados en una habitación

Cada sensor contiene a su vez una lista con las últimas actualizaciones que se ha producido en el mismo. Solamente se obtienen las 6 primeras actualizaciones de cada sensor. En el caso del software Z-Way, cuando se muestran todos los sensores en una misma ventana (Figura 6.13), es un poco complicado poder comparar los estados actuales de varios sensores relacionados entre sí (pueden mostrarse alejados unos de otros), además de hacer esta tarea imposible si se quiere comparar estados pasados entre ellos (se abre un *pop-up* (Figura 6.14) con el histórico del sensor, sin permitir la interacción con el resto de la ventana). Así, con el sistema web desarrollado este problema se elimina. Se permite al usuario mostrar los estados pasados de todos los sensores a la vez, mostrándose organizados en habitaciones (Figura 6.12).

Figura 6.13: Ventana de sensores en Z-Way

Si el usuario desea acceder a notificaciones más antiguas, se ha insertado una paginación. Esta paginación se ha implementado para reducir la cantidad de información que el servidor tiene que recuperar de la base de datos cada vez que se abre una habitación. De esta forma, se obtiene la información más útil al usuario (la información más reciente), permitiéndole a este acceder a información más antigua si es necesario. En este caso, la paginación se ha realizado con la ayuda del componente *Pagination* (Código 6.29), que proporciona algunas propiedades útiles para manejar el estado de la información que se muestra en cada sensor. Así, cada vez

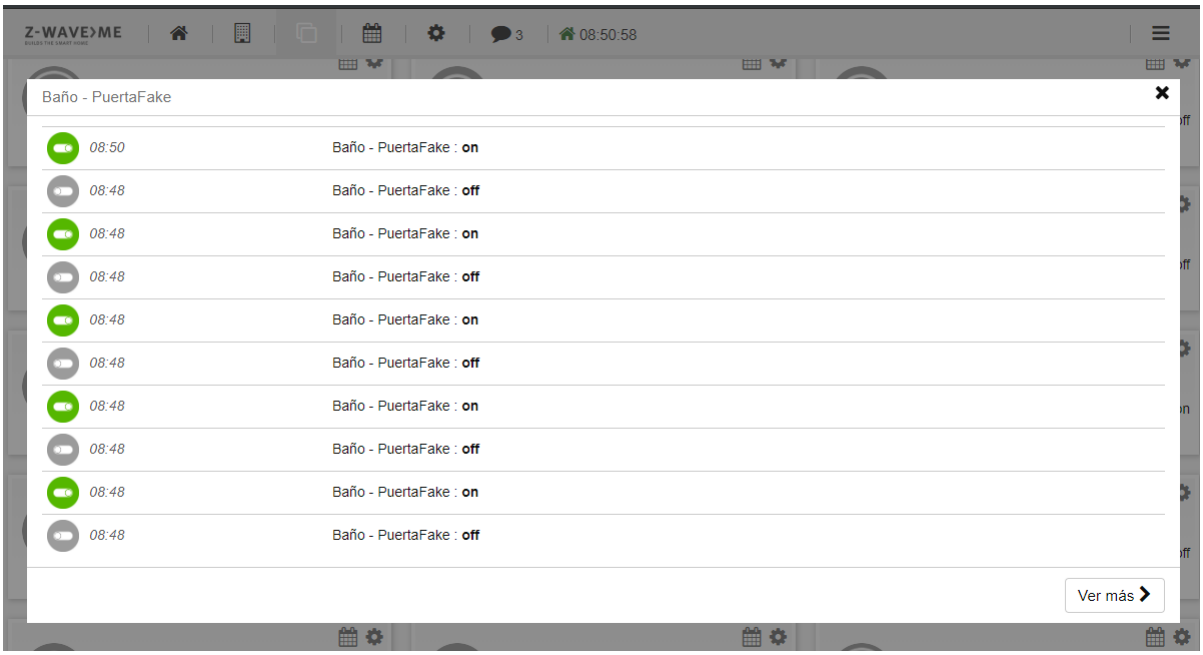


Figura 6.14: Pop-up Z-Way con el histórico de un sensor

que se cambie la *página*, se hará una consulta a la base de datos para obtener las notificaciones correspondientes.

Código 6.29: Componente Pagination

```
<Pagination
    activePage={this.state.sensores_mostrandose[key].pagina_actual}
    itemsCountPerPage={6}
    totalItemsCount={this.state.sensores_mostrandose[key].num_notificaciones}
    pageRangeDisplayed={3}
    onChange={this.cambiar_pagina_sensor.bind(this, { key })}
/>
```

6.3.6.4 Gestión de sensores

Los sensores del sistema tienen la posibilidad de ser *desactivados* por el usuario. En el caso de ser *desactivados*, no aparecen en la ventana principal y no se pueden incorporar a alguna habitación. Esta opción es necesaria porque puede darse la circunstancia de quitar un sensor de la vivienda. Si no se tiene esta opción, este sensor sigue almacenado pudiendo aparecer en alguna habitación. Al realizar esta funcionalidad, este sensor *desaparece* y no pueden realizarse acciones con él. En el caso que se quiera volver a incorporar, se permite volver a activar este sensor.

Para poder realizar estas acciones sobre los sensores, se muestra una lista con todos los sensores, ordenados por antigüedad creciente (Figura 6.15).

En esta lista, el elemento *Checkbox* muestra el estado del sensor. En caso de estar activo,

Figura 6.15: Lista de sensores que forman parte del sistema

el sensor se encuentra activo y visible en el resto del sistema. En caso contrario, se encuentra *silenciado*.

Además, se permite al usuario eliminar de forma definitiva un sensor del sistema.

6.3.6.5 Gestión de habitaciones

La vivienda en la cual se han incorporado los sensores contiene un total de 20 sensores instalados. Estos sensores de forma predeterminada no contienen un nombre fácilmente entendible para el usuario. Así, este usuario tiene que proporcionar un nombre que pueda identificar al sensor correctamente. Así, aunque el usuario proporcione un nombre adecuado (que mencione la ubicación concreta, magnitudes a medir, tipo...) no se pueden visualizar correctamente la información de estos, al estar todos los sensores mezclados entre sí (Figura 6.13). Por ello se han organizado en habitaciones.

En la ventana *Habitaciones* se tienen varias opciones implementadas hacer esto posible al usuario.

Añadir habitación En esta sección, el usuario puede añadir una nueva habitación, usando un formulario (Figura 6.16). El nombre de la habitación no puede ser repetido. En el caso de ser repetido, se muestra un aviso al usuario (Figura 6.17). Para realizar este aviso, se ha usado la función *M.toast()*, nativa de Materialize.

En este caso, al tratarse de una adición de nuevo contenido a la base de datos, se trata de una consulta POST; como se puede observar en el Código 6.30. Los parámetros que se envían se tratan de datos en formato JSON.

Código 6.30: JSON de la llamada POST

Figura 6.16: Formulario añadir nueva habitación

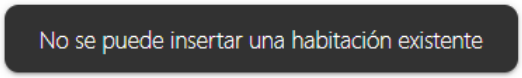


Figura 6.17: Notificación: nombre de habitación existente

```
fetch('/insertar_habitacion', {
    method: 'POST',
    body: JSON.stringify({
        habitacion: this.state.habitacion,
        imagen: this.state.archivo_insertado,
    }),
    headers: {
        Accept: 'application/json',
        'Content-Type': 'application/json',
    },
})
```

Lista habitaciones Otra operación disponible en el sistema web se trata de obtener un listado del nombre de las habitaciones (Figura 6.18).

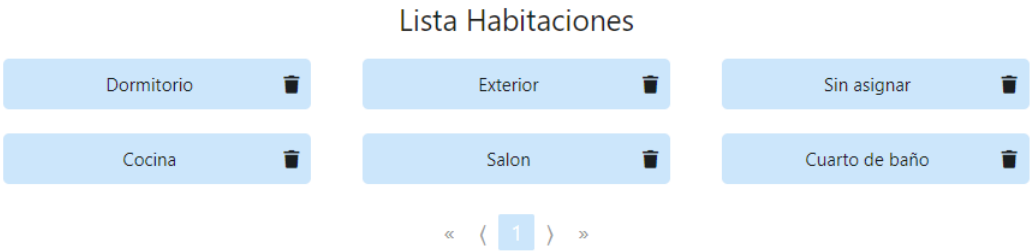


Figura 6.18: Listado de habitaciones

Esta opción se ha implementado para poder visualizar y eliminar las habitaciones que el usuario desee. Así, cuando este desea eliminar una habitación, debe confirmar la eliminación

en el mensaje de confirmación (Figura 6.19) que se muestra, usando el elemento *Confirm()* de JavaScript.

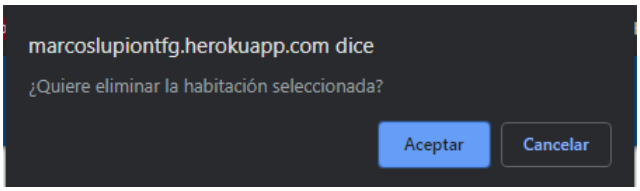


Figura 6.19: Pop-up eliminación de habitación

Asignación sensores a habitaciones En esta página se realiza una de las tareas fundamentales del sistema web, la situación de los sensores en las distintas habitaciones de la vivienda, con el objetivo de lograr una mayor contextualizacion de los datos procedentes de los sensores.

Para realizar esta asignación, se divide la página en un Combobox y dos columnas (Figura 6.20). Cada vez que se realiza una acción sobre las columnas, se realiza una consulta a la base de datos, por lo tanto, los datos de estas estarán en todo momento correspondidos con los datos que se tienen en la base de datos. De esta forma, aunque el usuario salga del sistema web, los cambios persistirán. En esta página web se pretende que en todo momento se refleje la información de la base de datos en las ventanas, así, cuando el usuario realiza cualquier acción, se registra para no perder ese contenido.

Figura 6.20: Ubicación de sensores en habitaciones

Lo elementos principales de esta sección se tratan de los siguientes.

- Combobox ”Seleccionar habitación”. En este componente (Figura 6.21) se muestran las habitaciones que se encuentran registradas en el sistema.

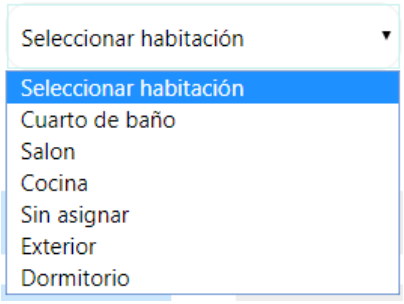


Figura 6.21: Combobox habitaciones almacenadas

- Situados. Cuando se selecciona una habitación en el combobox anterior, se muestran en esta columna los sensores que están ligados a la habitación seleccionada.

  Para eliminar un sensor de la habitación seleccionada, se hace clic sobre el sensor y automáticamente se desliga de la habitación, mostrándose en la columna que se menciona a continuación.

- Sin asignar. Se tratan de los sensores que no se encuentran ligados a alguna habitación. Los datos de los sensores de esta columna por tanto, no se muestran en la página principal del sistema.

  Así, para añadir un sensor a una habitación, basta con seleccionar el sensor de esta columna que se quiera añadir a la habitación seleccionada, mostrándose este en la lista *Situados* junto con el resto de sensores de la habitación.

6.3.7 Versión móvil/tablet

El sitio web desarrollado está adaptado para todo tipo de dispositivos móviles y tablets. Así, la funcionalidad es la misma se visualice en el dispositivo que el usuario desee. Durante el desarrollo del *front-end* en primer lugar se realizó enfocado a PC, sin embargo, se vio la necesidad de adaptarlo a dispositivos portátiles. Al realizar esta adaptación, no hubo que rehacer el trabajo que se había realizado anteriormente, sino que se tuvieron que definir estilos adaptados al tamaño de la ventana en la cual se muestra el sistema web.

El framework para el *front-end* que se usa se trata de Materialize. Como se explicó anteriormente, este framework proporciona un conjunto de clases CSS que se asignan a los elementos HTML que se visualizan en el navegador, proporcionandoles un estilo predefinido. De igual forma, este framework ofrece las herramientas necesarias para hacer adaptables los elementos a los distintos dispositivos.

Al usar Materialize, el *front-end* se divide en 12 columnas, todas ellas de igual tamaño. Así, los elementos pueden definirse para que ocupen las columnas que se desee. Para adaptar el *front-end* a versión móvil, a medida que se reduzca el tamaño, se tiene que aumentar el número de columnas que ocupa cada elemento para que los elementos que están situados dentro de este se visualicen correctamente. Además, Materialize proporciona un conjunto de *tags* que se pueden asignar en la clase de un elemento para definir el número de columnas que ocupa este

en cada tamaño de pantalla (Figura 6.22). Así, si un elemento está definido como *col s12 m6 l4 xl3*, cuando se tiene un dispositivo móvil, va a ocupar la pantalla completa, pero cuando se tiene un ordenador ocupará un cuarto de la pantalla, lográndose la adaptación del contenido correctamente.

Screen Sizes

	Mobile Devices <= 600px	Tablet Devices > 600px	Desktop Devices > 992px	Large Desktop Devices > 1200px
Class Prefix	`.s`	`.m`	`.l`	`.xl`
Container Width	90%	85%	70%	70%
Number of Columns	12	12	12	12

Figura 6.22: Dimensiones de diferentes dispositivos y clases de Materialize

Una vez se aprendió a usar este sistema de columnas con Materialize, la tarea de hacer el *front-end responsive*[5] fue tarea sencilla, se tuvo que dotar a cada elemento del sistema con un valor para cada tipo de tamaño de pantalla, logrando que se adapte adecuadamente.

En el Anexo 9 se muestran las que se corresponden con la versión móvil del sistema junto con su visualización correspondiente en la versión de ordenador.

6.4 Despliegue

El sistema web se desarrolló en modo local, haciendo uso de Git Bash para lanzar los comandos necesarios para la puesta a punto del servidor. Así, para ejecutar en modo local el sistema web se tiene que ejecutar simplemente el comando "*node src/index.js*". En el archivo *index.js* se encuentra la configuración de la base de datos, la localización del servidor y demás configuración necesaria para el funcionamiento del servidor.

Dado que el sistema web tiene que ser accesible desde la red, se usa Heroku para hacer el despliegue del mismo. Como ya se ha comentado en la Sección 3.5.1, Heroku se trata de un PaaS en el cual se despliegan aplicaciones de forma rápida y sencilla, proporcionando el repositorio en el que se encuentra la misma y especificando la tecnología que usa.

Como se puede observar, Heroku proporciona un panel de control (Figura 6.23) en el que se puede visualizar el estado de la aplicación. En este, se pueden observar los despliegues que se han realizado a lo largo del tiempo, así como información sobre la actividad y un enlace para abrir el sistema web.

[5]El sistema adapta su contenido a diferentes tamaños de pantalla, especialmente en dispositivos móviles.

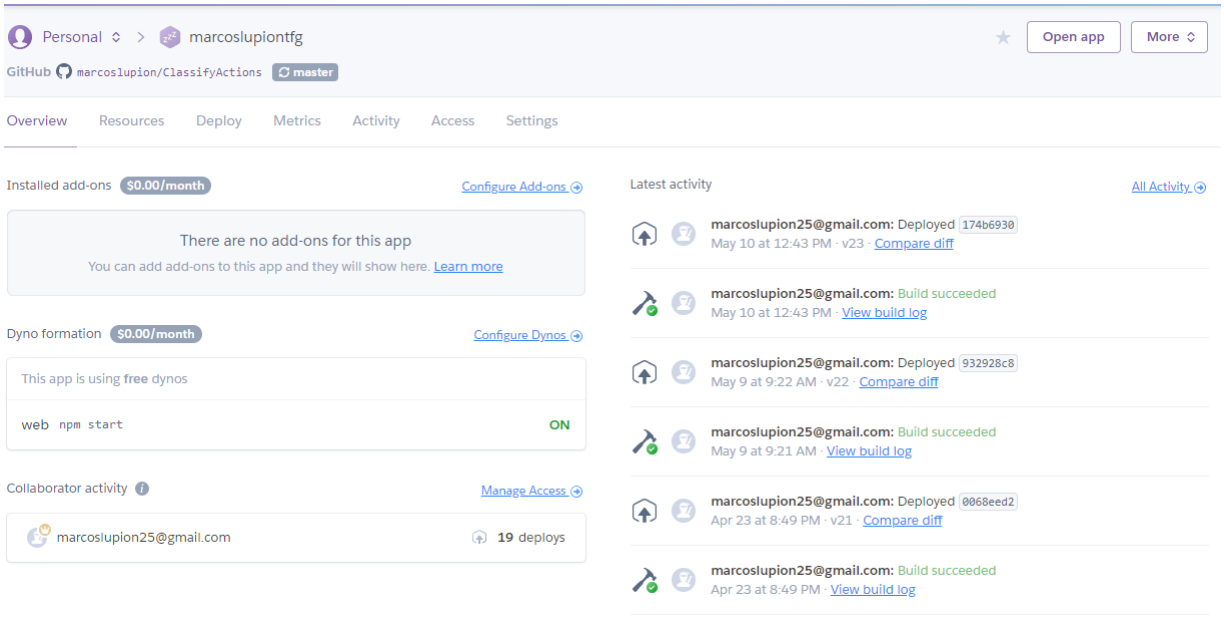


Figura 6.23: Panel de control en Heroku

Para realizar el despliegue de la aplicación, se realiza de forma automática (Figura 6.24) cada vez que se realiza una acción de *push*[6] sobre el repositorio, por lo que no hay que realizar un despliegue manual de la misma, y los cambios se aprecian rápidamente en la aplicación desplegada.

El sitio web se puede encontrar en la dirección `https://marcoslupiontfg.herokuapp.com`

[6]Corresponde con la acción de subir los cambios al repositorio

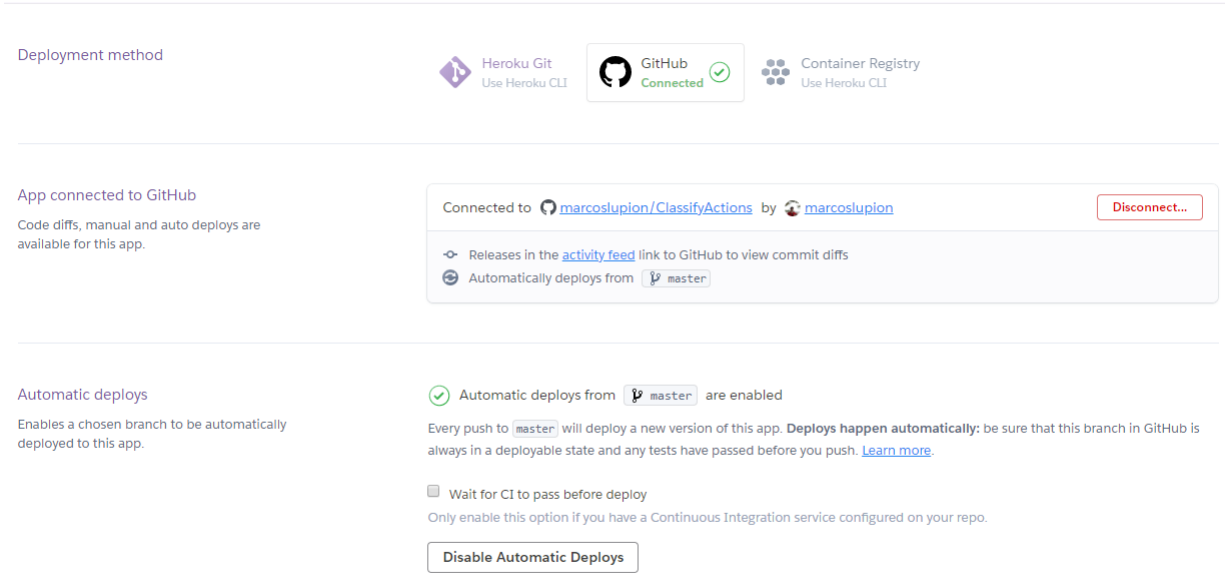


Figura 6.24: Despliegue automático

7 Resultados, conclusiones y trabajo futuro

7.1 Resultados

En el trabajo se ha implementado un sistema IoT de monitorización de sensores situados en una vivienda. El sistema proporciona una interfaz web accesible a través de Internet para poder acceder desde cualquier lugar, de esta forma siempre es posible controlar el estado de los sensores. El sistema se trata de un sistema IoT, ya que se recogen datos del mundo físico a través de sensores y se muestran en una plataforma web. Para realizar el sistema IoT completo, el trabajo se dividió en dos subsistemas.

El primer subsistema consiste en un sistema IoT en la niebla o neblina. En este, se realiza la obtención de datos procedentes de los sensores por parte de un dispositivo Raspberry. Así, estos datos son procesados en este dispositivo, realizando tareas de computación en la niebla o neblina como la visualización de estos datos en un software open source proporcionado o el tratamiento de estos para poder utilizarlos posteriormente.

A continuación, se implementó un sistema IoT en la nube. Este sistema obtiene los datos almacenados en el dispositivo Raspberry y los almacena en una base de datos en la nube. Además, proporciona un sistema web en el cual se permite la visualización de estos datos y la distribución de estos datos en diferentes ventanas con el objetivo de facilitar la comprensión de la información que se muestra.

Ambos subsistemas definidos interactúan entre sí proporcionando al usuario una herramienta de visualización y gestión de los datos proporcionados por los sensores situados en la vivienda.

7.2 Conclusiones

El sistema IoT desplegado está disponible en todo momento y accesible desde cualquier lugar, requisitos básicos de este, ya que el software open source que se usa solamente permite visualizar los datos de los sensores accediendo al dispositivo Raspberry. Además, se permite realizar un tratamiento de los datos procedentes de los sensores para poder usarlos en las aplicaciones que el usuario desee, en este caso, en un sistema web. El sistema IoT desplegado, en definitiva, permite obtener una monitorización de los datos de los sensores de la vivienda mediante un sistema web propio, sin tener que hacer uso del software proporcionado. De esta forma, se gana flexibilidad en la manipulación de los datos y se amplían los posibles usos de estos datos, ya que pueden usarse para realizar tareas de *big data* o *machine learning* con estos.

Para realizar el trabajo, se tuvieron que seguir una serie de etapas.

En primer lugar, se realizó un estudio sobre la teoría de Internet de las Cosas y las *smart homes*, ámbitos en los que se incluye el trabajo. Este estudio no fue realizado sin tener conocimiento previo de ambos campos. En las asignaturas de *Tecnologías de Acceso a Red* y *Fundamentos de redes de computadoras* se realizó una introducción sobre estos temas, ya que están íntimamente relacionados con las redes de computadoras.

Una vez que se tenía una base sobre ambos temas y sus posibles aplicaciones en la vida real, se procedió a realizar el estudio y selección de elementos hardware a incorporar en el sistema. El conocimiento obtenido a través de las asignaturas *Fundamentos de electrónica* fue la base para escoger los sensores en base a las características del sistema. Además de escoger los sensores, se tuvo que elegir el protocolo de comunicación entre estos. Esta tarea no fue tan complicada porque se tenían conocimientos proporcionados en la asignatura *Transmisión de datos y redes de computadoras*. Una vez obtenidos estos dispositivos hardware, se procedió a realizar la conexión y configuración de estos. Concretamente, el dispositivo Raspberry fue puesto en funcionamiento ya que previamente se había usado en la asignatura *Arquitectura de computadoras*.

Una vez se implantó el sistema hardware formado por los sensores y el controlador, se diseñó un sistema web. Este diseño se realizó siguiendo las pautas de las asignaturas *Ingeniería de requisitos*, *Modelado del software* y *Bases de datos*. Una vez diseñado el sistema, se procedió a implementarlo. Para realizar la programación web se hizo uso de conocimientos obtenidos en *Tecnologías web* y *Tecnologías Multimedia*.

El sistema en las primeras fases fue puesto en funcionamiento en la máquina local. Una vez comprobado el correcto funcionamiento de este, se procedió a desplegarlo en la nube. Para realizar estas tareas, las asignaturas de *Sistemas operativos*, *Administración de redes y sistemas operativos* y *Seguridad informática* aportaron las nociones básicas sobre el uso y despliegue de aplicaciones en servicios de la nube.

En conclusión, a partir de los conceptos aprendidos en las asignaturas del grado, ha sido posible realizar la implementación de un sistema de monitorización de una vivienda. Este sistema ha servido para comprender los conceptos de Internet de las Cosas y de *smart homes*. Así, una vez implementado el sistema y funciona correctamente, es posible incorporar funcionalidad adaptada al usuario a medida que se desee.

7.3 Trabajo futuro

El sistema que se ha implementado permite al usuario la monitorización de la vivienda cuando este lo desee. Sin embargo, la información que se muestra al usuario se trata solamente de datos obtenidos mediante los sensores.

Las *smart homes*, a parte de visualizar la información de los sensores, permiten la ejecución de actuaciones en base a los datos que proporcionan estos. Sin embargo, en este trabajo no se ha dispuesto de actuadores para poder realizar este tipo de acciones. Como trabajo futuro, se podría realizar la incorporación de dispositivos de este tipo al sistema, con el objetivo de actuar cuando ocurren ciertas condiciones en la vivienda.

Por otra parte, la información de la que dispone el usuario consiste en datos proporcionados por los sensores, indicando el estado en el que se encuentran. Esta información es útil consultarla en caso de comprobar diferentes hechos que han ocurrido, como por ejemplo, a qué hora llegó un usuario a la vivienda o cuánto tiempo ha estado durmiendo una persona. Sin embargo, al no visualizarse la información de forma gráfica, el usuario puede perderse y no hacer uso de ella. Así, como trabajo futuro se propone incorporar gráficos que muestren esta información de forma clara y concisa.

Actualmente, el aprendizaje automático está muy presente en la vida de las personas, ya que estas generan una gran cantidad de datos diariamente y existen servicios que hacen uso de estos datos para extraer información a partir de ellos y realizar servicios automatizados en base a patrones que siguen estos datos (Alsheikh y cols., 2014) (Kumar y cols., 2019). El uso de esta información va desde la muestra de publicidad personalizada o recomendaciones en base a los gustos. Así, en un futuro se pretende incorporar un algoritmo de *machine learning* capaz de procesar los datos que generan los sensores con el objetivo de reconocer las actividades que realiza un usuario en la vivienda. Por lo tanto, el sistema no solamente mostrará los datos de los sensores, sino que los procesa extrayendo conclusiones de lo que ocurre en la vivienda en tiempo real (Samarah y cols., 2017). De esta forma, se podrán conocer las actividades que realizan los usuarios en la vivienda. Esta información es importante en el caso de que en la vivienda habiten personas mayores, permitiendo un mayor control sobre la actividad que realizan y detectar posibles problemas.

En conclusión, se ha implementado un sistema que se trata de la base para desarrollar un ambiente inteligente completamente adaptado al usuario.

Bibliografía

Alsheikh, M. A., Lin, S., Niyato, D., y Tan, H.-P. (2014). Machine learning in wireless sensor networks: Algorithms, strategies, and applications. *IEEE Communications Surveys Tutorials*, *16*(4), 1996-2018.

Chana, M., Campoa, E., Estèvea, D., y Fourniols, J.-Y. (2009). Smart homes — current features and future perspectives. *Maturitas*, *64*(2), 90-97.

Cisco. (2018-2019). *Iot fundamentals: Connecting things*. Autor.

daCosta, F., y Henderson, B. (2013). *Rethinking the internet of things: A scalable approach to connecting everything*. Apress.

de Valladolid, U. (2019). *Sensores.* Descargado de `http://www.isa.cie.uva.es/~maria/sensores.pdf`

Ding, D., Cooper, R. A., Pasquina, P. F., y Fici-Pasquina, L. (2011). Sensor technology for smart homes. *Maturitas*, *69*(2), 131-136.

Ding, D., Cooper, R. A., Pasquina, P. F., y Fici-Pasquina, L. (2016). Smart homes and home health monitoring technologies for older adults: A systematic review. *International Journal of Medical Informatics*, *91*(2), 44-59.

Electronics, A. (2019). *Data sheet - arun pm3.* Descargado de `http://www.arun-electronics.co.uk/pdf/Pressure%20Mat%20Data%20Sheet%20-%20issue%201.pdf`

Fibaro. (2019a). *Documentación - door/window sensor.* Descargado de `https://manuals.fibaro.com/content/manuals/en/FGK-10x/FGK-10x-EN-T-v2.0.pdf`

Fibaro. (2019b). *Documentación - door/window sensor 2.* Descargado de `https://manuals.fibaro.com/content/manuals/en/FGDW-002/FGDW-002-EN-T-v1.1.pdf`

Fibaro. (2019c). *Documentación - motion sensor fibaro.* Descargado de `https://manuals.fibaro.com/content/manuals/en/FGMS-001/FGMS-001-EN-T-v2.1.pdf`

Foundation, R. P. (2019). *Documentación raspberry pi 3 b+.* Descargado de `https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf`

Geng, H. (2016). *Internet of things and data analytics handbook*. John Wiley Sons, Inc.

Gentry, T. (2009). Smart homes for people with neurological disability: state of the art. *NeuroRehabilitation*, *25*(3).

Ghayvat, H., y Mukhopadhyay, S. C. (2017). *Wellness protocol for smart homes : An integrated framework for ambient assisted living.* Springer.

Giamas, A. (2019). *Mastering mongodb 4.x - second edition.* Packt Publishing.

Hanes, D., Salgueiro, G., y Barton, R. (2017). *Iot fundamentals: Networking technologies, protocols, and use cases for the internet of things.* Cisco Press.

Hersent, O., Boswarthick, D., y Elloumi, O. (2011). *The internet of things: Key applications and protocols.* Wiley.

Hoof, J., Demiris, G., Wouters, E. J. M., y van Hoof, J. (2016). *Handbook of smart homes, health care and well-being.* Springer International Publishing AG.

Kereki, F. (2018). *Modern javascript web development cookbook.* Packt Publishing.

Koroliova, E. (2018). *Mern quick start guide.* Packt Publishing.

Kumar, D. P., Amgoth, T., y Annavarapu, C. S. R. (2019). Machine learning algorithms for wireless sensor networks: A survey. *Information Fusion*, *49*, 1-25.

Lee, J.-S., Su, Y.-W., y Shen, C.-C. (2007). A comparative study of wireless protocols: Bluetooth, uwb, zigbee, and wi-fi. *IECON 2007 - 33rd Annual Conference of the IEEE Industrial Electronics Society*, 46-51.

Lengstorf, J., y Leggetter, P. (2013). *Realtime web apps.* Apress.

Materialize. (2019). *Documentación.* Descargado de `https://materializecss.com/getting-started.html`

MongoDB. (2019). *Documentación.* Descargado de `https://docs.mongodb.com/manual/`

Paetz, C. (2017). *Z-way essentials.* Descargado de `https://z-wave.me/essentials`

Prabhu, A., y Shenoy, A. (2016). *Introducing materialize.* Apress L. P.

Rao, M. (2018). *Internet of things with raspberry pi 3.* Packt Publishing.

Ravulavaru, A. (2017). *Practical internet of things with javascript.* Packt Publishing.

ReactJS. (2019). *Documentación.* Descargado de `https://reactjs.org/docs/getting-started.html`

Samarah, S., Zamil, M. G. A., Aleroud, A. F., Rawashdeh, M., Alhamid, M. F., y Alamri, A. (2017). An efficient activity recognition framework: Toward privacy-sensitive health data sensing. *The Journal of the Acoustical Society of America*, *117*(3), 988-988.

Schwartz, M. (2016). *Building smart homes with raspberry pi zero: build revolutionary and incredibly useful home automation projects with the all-new pi zero.* PACKT Publishing.

Shumei, Z., McCullagh, P., Zheng, H., y Nugent, C. (2017). Situation awareness inferred from posture transition and location: Derived from smartphone and smart home sensors. *IEEE Transactions on Human-Machine Systems*, *47*(6), 814-821.

Shute, Z. (2019). *Advanced javascript.* Packt Publishing.

Vasan, S. (2017). *Pro mern stack : Full stack web app development with mongo, express, react, and node.* Apress L. P.

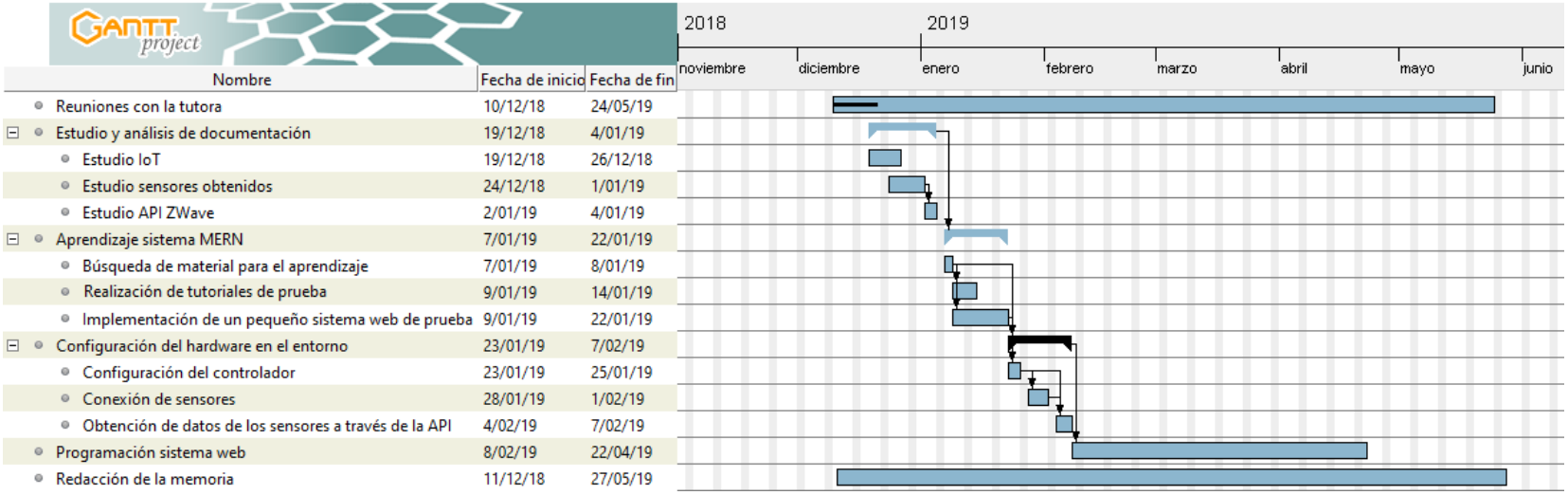
Yin, J., Fang, M., Mokhtari, G., y Zhang, Q. (2016). Multi-resident location tracking in smart home through non-wearable unobtrusive sensors. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *9677*(6), 3-13.

ZWave.me. (2019). *Z-way api.* Descargado de `https://zwayhomeautomation.docs.apiary.io/#`

8 Anexo I. Diagramas de Gantt

En este anexo se muestran los diagramas de Gantt asociados al proyecto. En primer lugar, se muestra el diagrama que muestra la planificación inicial de tareas y tiempos de duración. En segundo y último lugar se visualiza el diagrama de Gantt que contiene la ejecución de las tareas a lo largo del tiempo y su duración asociada.

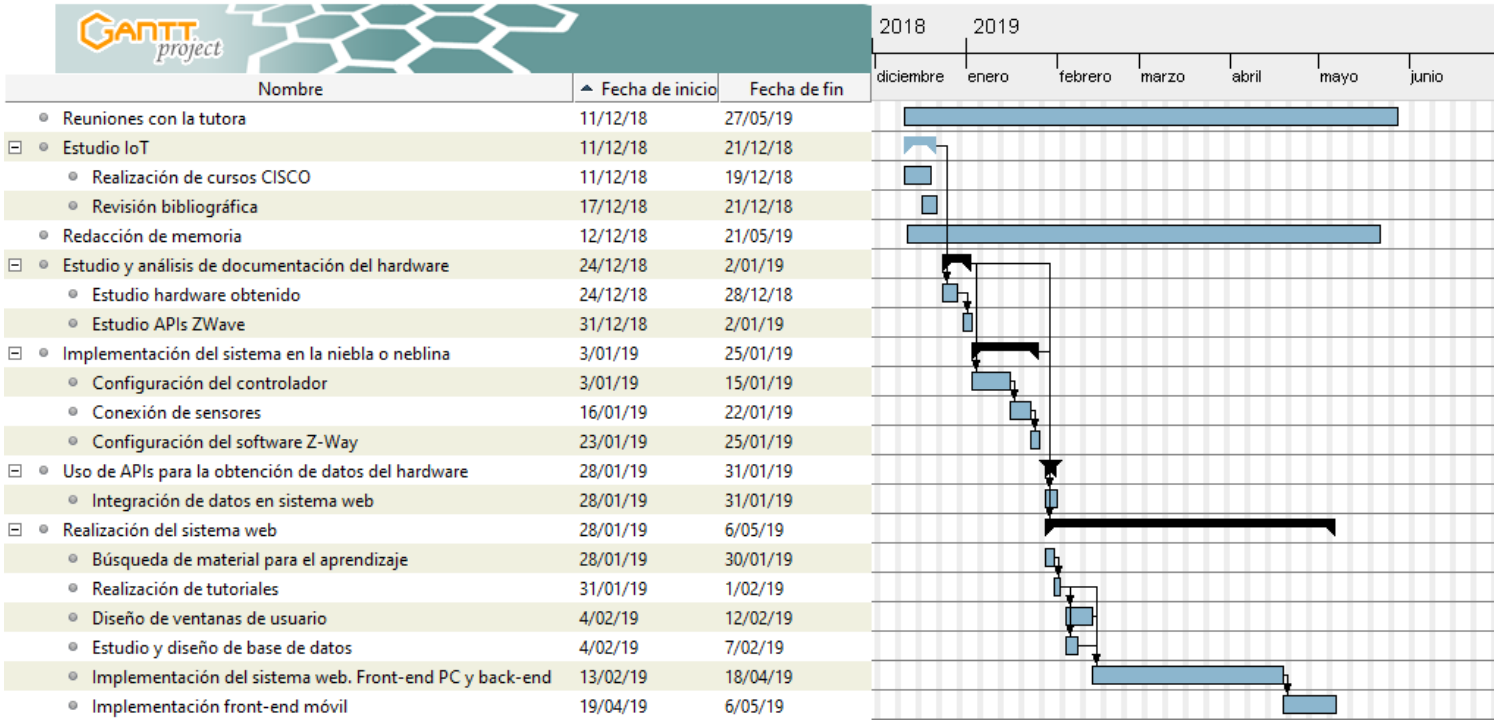
8.1 Planificación



Nombre	Fecha de inicio	Fecha de fin
Reuniones con la tutora	10/12/18	24/05/19
Estudio y análisis de documentación	19/12/18	4/01/19
Estudio IoT	19/12/18	26/12/18
Estudio sensores obtenidos	24/12/18	1/01/19
Estudio API ZWave	2/01/19	4/01/19
Aprendizaje sistema MERN	7/01/19	22/01/19
Búsqueda de material para el aprendizaje	7/01/19	8/01/19
Realización de tutoriales de prueba	9/01/19	14/01/19
Implementación de un pequeño sistema web de prueba	9/01/19	22/01/19
Configuración del hardware en el entorno	23/01/19	7/02/19
Configuración del controlador	23/01/19	25/01/19
Conexión de sensores	28/01/19	1/02/19
Obtención de datos de los sensores a través de la API	4/02/19	7/02/19
Programación sistema web	8/02/19	22/04/19
Redacción de la memoria	11/12/18	27/05/19

Figura 8.1: Planificación - Diagrama de Gantt

8.2 Ejecución



Nombre	Fecha de inicio	Fecha de fin
Reuniones con la tutora	11/12/18	27/05/19
Estudio IoT	11/12/18	21/12/18
Realización de cursos CISCO	11/12/18	19/12/18
Revisión bibliográfica	17/12/18	21/12/18
Redacción de memoria	12/12/18	21/05/19
Estudio y análisis de documentación del hardware	24/12/18	2/01/19
Estudio hardware obtenido	24/12/18	28/12/18
Estudio APIs ZWave	31/12/18	2/01/19
Implementación del sistema en la niebla o neblina	3/01/19	25/01/19
Configuración del controlador	3/01/19	15/01/19
Conexión de sensores	16/01/19	22/01/19
Configuración del software Z-Way	23/01/19	25/01/19
Uso de APIs para la obtención de datos del hardware	28/01/19	31/01/19
Integración de datos en sistema web	28/01/19	31/01/19
Realización del sistema web	28/01/19	6/05/19
Búsqueda de material para el aprendizaje	28/01/19	30/01/19
Realización de tutoriales	31/01/19	1/02/19
Diseño de ventanas de usuario	4/02/19	12/02/19
Estudio y diseño de base de datos	4/02/19	7/02/19
Implementación del sistema web. Front-end PC y back-end	13/02/19	18/04/19
Implementación front-end móvil	19/04/19	6/05/19

Figura 8.2: Ejecución - Diagrama de Gantt

9 Anexo II. Sistema web. Ventanas de usuario.

En este anexo se muestran las ventanas del sistema web. Para cada ventana de usuario, se muestra la ventana que se visualiza en un ordenador y la correspondiente ventana en un dispositivo móvil.

9.1 Inicio de sesión

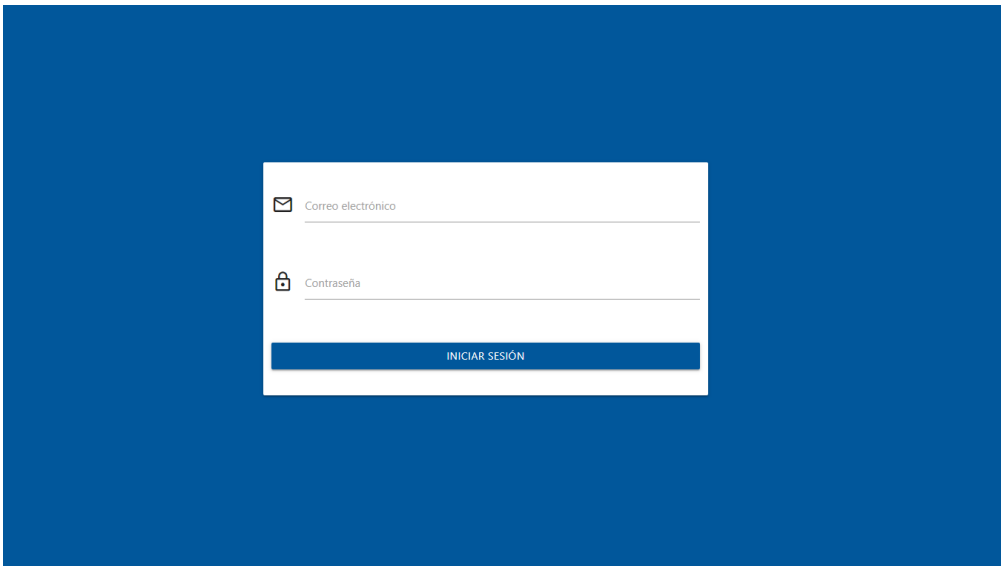


Figura 9.1: Inicio Sesión - PC

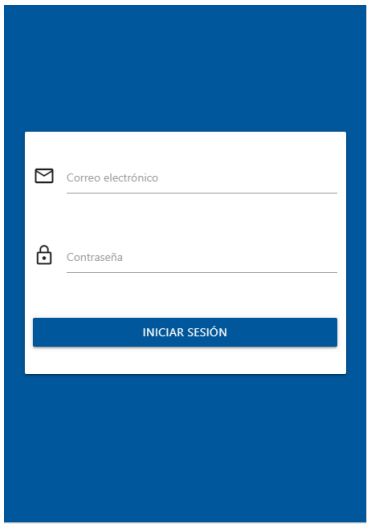


Figura 9.2: Inicio Sesión - Móvil

9.2 Cambiar contraseña

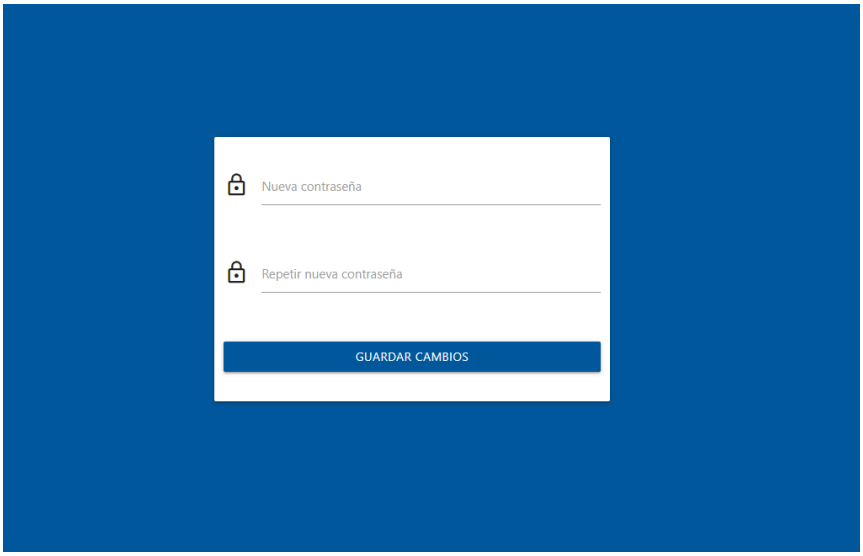


Figura 9.3: Cambiar contraseña - PC

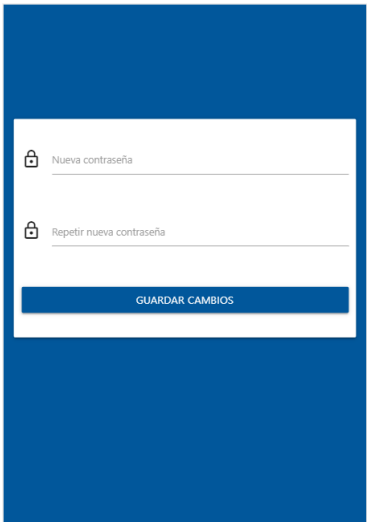


Figura 9.4: Cambiar contraseña - Móvil

9.3 Página principal

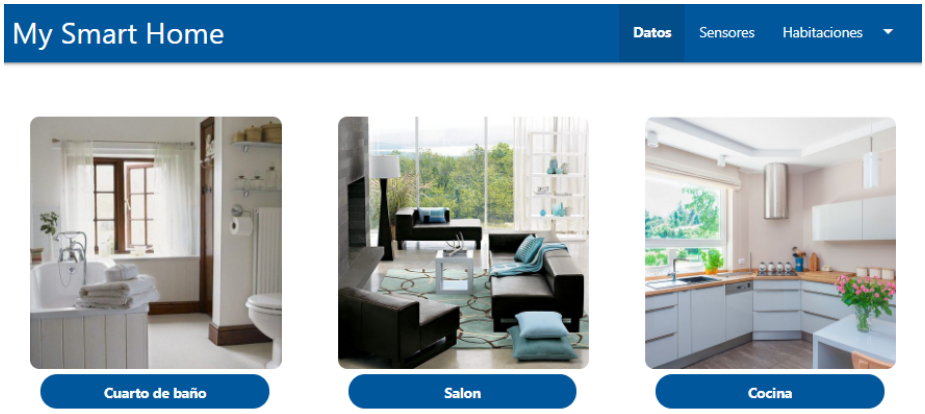


Figura 9.5: Principal - PC

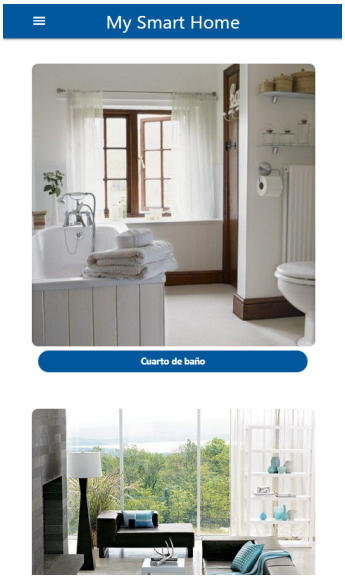


Figura 9.6: Principal - Móvil

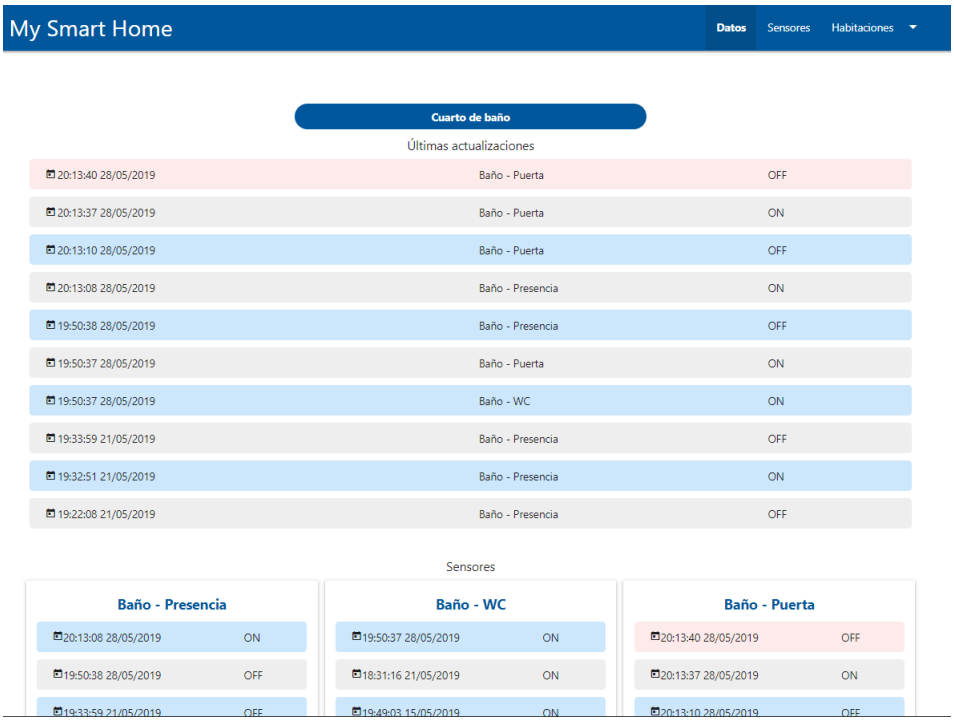


Figura 9.7: Principal - Habitación seleccionada - PC

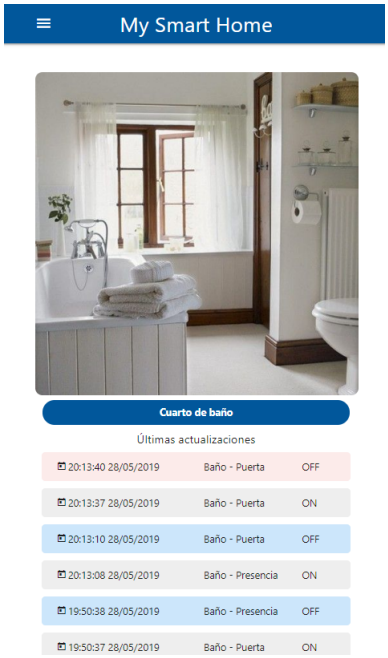


Figura 9.8: Principal - Habitación seleccionada - Móvil

9.4 Habitaciones - Asignar-Desasignar sensores

Figura 9.9: Habitaciones - Asignar-Desasignar sensores - PC

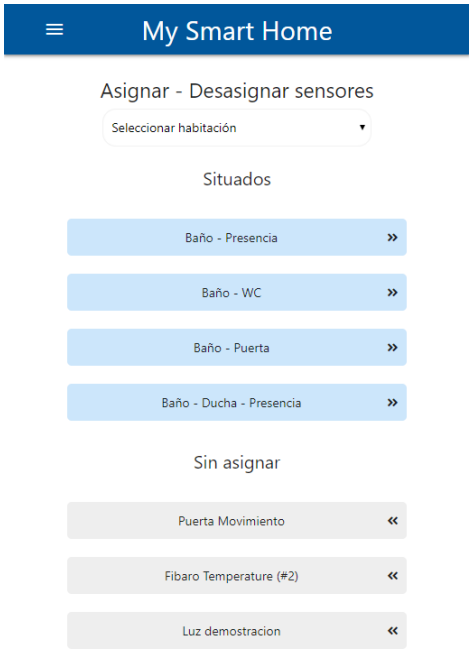


Figura 9.10: Habitaciones - Asignar-Desasignar sensores - Móvil

9.5 Habitaciones - Añadir habitación

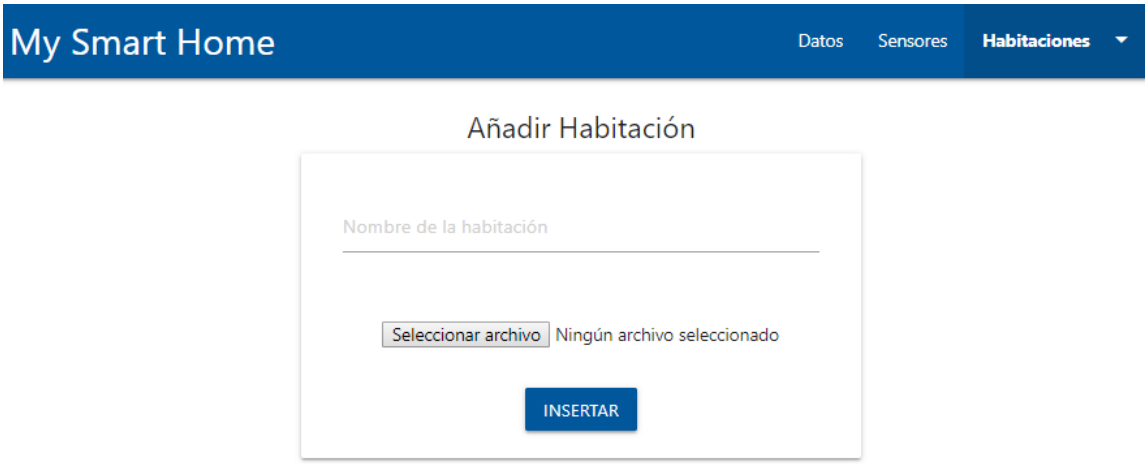


Figura 9.11: Habitaciones - Añadir - PC

Figura 9.12: Habitaciones - Añadir - Móvil

9.6 Habitaciones - Lista habitaciones

Figura 9.13: Habitaciones - Lista - PC

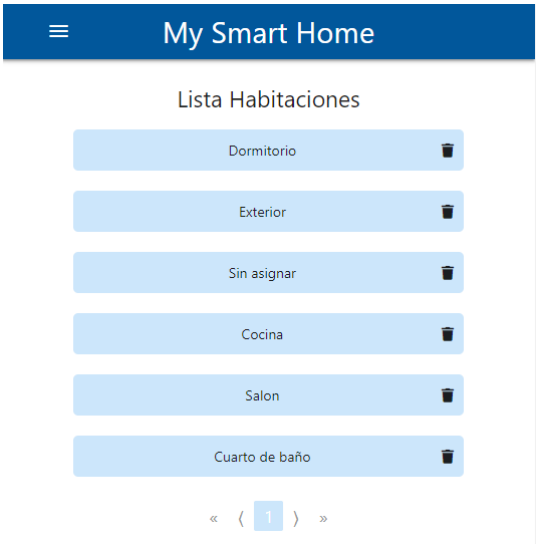


Figura 9.14: Habitaciones - Lista - Móvil

9.7 Sensores - Lista

Figura 9.15: Sensores - PC

Figura 9.16: Sensores - Móvil

UNIVERSIDAD DE ALMERÍA

Resumen/Abstract

En la actualidad, cada vez es mayor el número de objetos de la vida cotidiana conectados a Internet, proporcionando información sobre el medio en el que se encuentran y ejecutando acciones en base a esta. Esto es lo que se conoce como Internet de las Cosas (IoT).

Concretamente, en este proyecto se construye una infraestructura de sistema IoT en una vivienda, convirtiéndola en una Smart Home. En la vivienda, se ubican diferentes sensores que se configuran para comunicarse inalámbricamente a controladores, que posteriormente envían la información proporcionada por los sensores a un servidor web. A través de dicho sistema web desplegado en Internet, el usuario puede realizar una monitorización del estado de la vivienda cuando y donde desee.

En este documento se explica cómo se ha implementado el sistema IoT, desde la ubicación y configuración de los sensores hasta el despliegue del sistema web.

Nowadays, the number of everyday objects connected to the Internet is increasing, providing information about the environment in which they are located and executing actions based on this information. This is what is known as the Internet of Things (IoT).

Specifically, in this project, an IoT system infrastructure is built in a house, converting it into a Smart Home. In that house, different sensors are configured to communicate wirelessly to controllers that subsequently send the information provided by the sensors to a web server. Through this web system deployed on the Internet, the user can monitor the state of the house whenever and wherever he wishes.

This document explains how the IoT system has been implemented, from the location and configuration of the sensors to the deployment of the web system.

GRADO EN INGENIERÍA INFORMÁTICA, 2018/2019