UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

IMPLEMENTACIÓN Y EVALUACIÓN DE PLATAFORMAS EN LA NUBE PARA SERVICIOS DE IoT

MIRIAM ORTIZ MONET

TUTOR: NARCÍS CARDONA MARCET

DIRECTOR: GERARDO MARTÍNEZ PINZÓN

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2018-19

Valencia, 10 de septiembre de 2019

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
www.etsit.upv.es

VLC/ CAMPUS VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS Gestión ambiental verificada REG.NO. ES-CV- 000030

Resumen

En este Trabajo de Fin de Grado, se ha desarrollado una plataforma en la nube para servicios de Internet de las Cosas, cuyo objetivo es ofrecer una solución de bajo coste desarrollada con herramientas de código abierto (*open source*). Para ello, previamente se ha llevado a cabo un estudio de las diferentes soluciones disponibles para cada uno de los elementos que conforman la arquitectura de este sistema, comparando sus prestaciones y características principales, con el fin de hacer uso de las herramientas óptimas para una plataforma IoT. Posteriormente se han llevado a cabo la implementación del sistema haciendo uso de las herramientas seleccionadas. Para su validación se ha verificado tanto el cumplimiento de la función prevista para cada uno de los nodos como el funcionamiento del sistema al completo, llegando a visualizar los datos inicialmente recogidos por el sensor a través de la aplicación web que se ha desarrollado y se pone a disposición del usuario final de la plataforma.

Resum

En aquest Treball de Fi de Grau, s'ha desenvolupat una plataforma en el núvol per a serveis d'Internet de les Coses, l'objectiu és oferir una solució de baix cost desenvolupada amb eines de codi obert (open source). Per a això, prèviament s'ha dut a terme un estudi de les diferents solucions disponibles per a cada un dels elements que conformen l'arquitectura d'aquest sistema, comparant les seves prestacions i característiques principals, per tal de fer ús de les eines òptimes per a una plataforma IoT. Posteriorment s'han dut a terme la implementació del sistema fent ús de les eines seleccionades. Per a la seva validació s'ha verificat tant el compliment de la funció prevista per a cada un dels nodes com el funcionament del sistema al complet, arribant a visualitzar les dades inicialment recollides pel sensor a través de l'aplicació web que s'ha desenvolupat i es posa a disposició de l'usuari final de la plataforma.

Abstract

In this End of Degree Project, a cloud platform for Internet of Things services has been developed, whose goal is to offer a low-cost solution developed with open source tools. To this end, a study of the different solutions available for each of the elements that make up the architecture of this system has previously been carried out, comparing its main features and capabilities, in order to make use of the optimal tools for an IoT platform. Subsequently, the system has been implemented using the selected tools. For its validation, both the fulfillment of the function foreseen for each of the nodes and the operation of the entire system have been verified, visualizing the data initially collected by the sensor through the web application that has been developed and made available to the end user of the platform.

Índice general

1. Introducción **3**

1.1. Objetivos 3
1.2. Estructura de la memoria 4
1.3. Distribución de tareas 4

2. Estado del arte **6**

2.1. Internet de las cosas 6
2.1.1. Capacidades de un sistema IoT 6
2.1.2. Arquitectura de un sistema IoT 7
2.1.3. Cloud Computing para servicios de IoT 10
2.1.3.1. Arquitectura de Cloud Computing 10
2.1.3.2. Modelos de Servicio de Cloud Computing 11
2.1.3.3. Modelos de implementación de Cloud Computing 12
2.1.3.4. Plataformas de Cloud Computing para IoT 12
2.1.3.4.1. Plataformas de Cloud Computing Open Source 12
2.1.3.4.2. Plataformas de Cloud Computing No Open Source 15

3. Implementación de una Plataforma en la Nube Privada para servicios de IoT **19**

3.1. Arquitectura del sistema 19
3.1.1. Sistema del nodo cliente 19
3.1.2. Broker MQTT 23
3.1.3. Sistema del nodo servidor 24
3.1.3.1. Servidor 24
3.1.3.1.1. Servidor Web 24
3.1.3.1.2. Base de Datos MySQL 29
3.1.3.1.3. Aplicación Web 31
3.1.3.2. Cliente MQTT 44
3.1.4. Usuarios finales 44

4. Validación de la Plataforma en la Nube Privada para servicios de IoT 45

5. Conclusiones y líneas de trabajo futuras 51

5.1. Conclusiones 51

5.2. Líneas de trabajo futuras.....52

Bibliografía 53

Anexos 55

Anexo I: Datos Técnicos del Sensor de Temperatura y Humedad DHT11 55

Anexo II: Recogida de Datos y Publicación de Cliente MQTT 55

Anexo III: Subscripción de Cliente MQTT y Almacenamiento en Base de Datos 56

Anexo IV: Fragmentos de código de la Aplicación Web.....57

1. Introducción

En los últimos años, Internet de las Cosas (IoT, *Internet of Things*, en inglés) está cobrando cada vez mayor presencia en diferentes y múltiples campos como son: agricultura y ganadería, ciudades inteligentes, industria, ... Es por ello por lo que se ha convertido en un importante tema de investigación.

El concepto de IoT se basa en la interconexión e integración del mundo físico con el mundo de la información, en otras palabras, la interconexión masiva de dispositivos y objetos a la red. Esto genera un enorme volumen de datos, por lo que realizar una gestión eficiente de los mismos permitirá conseguir el desarrollo de aplicaciones y servicios que mejoren la calidad de vida de la sociedad.

Todo sistema IoT debe contar con una serie de capacidades que se corresponden básicamente con los elementos que componen dicho sistema. Estas capacidades consisten en identificar los objetos de la red, captar la información requerida de dichos objetos, comunicar inalámbricamente los nodos que conforman el sistema, procesar los datos recolectados y por último gestionar e interpretar los datos procesados con el fin de ofrecer diferentes servicios al usuario final. En cuanto a la arquitectura de estos sistemas, pueden presentarse varios modelos, como se verá en el apartado 2.1.2. de este documento, siendo la arquitectura de cinco capas la más utilizada, en ella se diferencian las capas de percepción, red, procesado, aplicación y negocio.

Debido a las grandes cantidades de información que genera Internet de las Cosas, surge su relación con la computación en la nube (*Cloud Computing*, en inglés). Respecto a esta, se explicará tanto su arquitectura como los diferentes modelos de servicio que ofrece (SaaS, PaaS, IaaS) y modelos de implementación que pueden existir (nube pública, comunitaria, privada e híbrida).

La combinación de IoT con la computación en la nube permite el almacenamiento, procesado y acceso de los datos, así como su monitorización y análisis para ofrecer servicios a todo tipo de aplicaciones y consiguiendo minimizar los problemas de memoria y espacio que se presentan al manejar tal cantidad de información. Esto puede suponer una verdadera revolución industrial debido que se puede conseguir automatizar tareas mejorando así los procesos de ejecución, tiempos de producción y costes. Por otro lado, puede llevar al surgimiento de nuevos modelos de negocio tanto a nivel local como global, en ámbitos tan dispares como, por ejemplo; educación, comunicaciones, ciencia...

Cada vez surgen en el mercado más plataformas de computación en la nube destinadas a servicios de IoT, las cuales integran diferentes soluciones y tecnologías propias o incluso de código abierto (plataformas *open source*), algunas de estas se presentarán en el apartado 2.1.3.4. del documento.

Por estas razones, en este Trabajo Final de Grado se ha propuesto diseñar, implementar y evaluar una plataforma en la nube privada orientada a servicios de IoT cumpliendo los requisitos de que sea de bajo coste y su implementación se lleve a cabo con herramientas *open source*.

1.1. Objetivos

Este trabajo final de grado presenta como objetivo fundamental la implementación y posterior evaluación de una plataforma en la nube privada, orientada a servicios de Internet de las Cosas.

Hoy en día se puede encontrar una amplia gama de plataformas IoT de pago que integran tecnologías para soluciones IoT, por ello, el alcance propuesto en este TFG es conseguir que la plataforma en la nube cumpla con dos requisitos primordiales: bajo coste y desarrollo con herramientas de código abierto. Para conseguirlo se han definido varios puntos a cumplir:

- Realizar una extensa revisión bibliográfica que permita identificar las principales soluciones de plataformas en la nube, y herramientas de desarrollo para servicios de IoT.
- Definir la arquitectura del sistema IoT a desarrollar, identificando con claridad cada uno de los elementos que la componen.
- Implementar el nodo cliente haciendo uso de un sensor que será el responsable de tomar las medidas, conectándolo a un ordenador de placa reducida como es Raspberry Pi.
- Desarrollar un nodo servidor en el que se almacenen los datos, teniendo en cuenta que deberá soportar diferentes servicios de IoT.
- Identificar e implementar un protocolo de comunicación a usar entre el nodo cliente y servidor con el fin de poder enviar los datos recogidos y almacenarlos en la base de datos.
- Diseñar y desarrollar una aplicación web desde la cual el usuario final de esta plataforma sea capaz de visualizar en tiempo real los datos que han sido recogidos por los sensores en el período de tiempo que desee seleccionar, así como estadísticas de estos.

1.2. Estructura de la memoria

En los siguientes puntos se especifican la organización y distribución del contenido de este documento:

- **Estado del arte:** Se expone el estado del arte de Internet de las Cosas detallando las capacidades y los diferentes modelos de arquitectura que puede presentar un sistema IoT. A su vez se hace un estudio de las plataformas de Cloud Computing para servicios IoT que se pueden encontrar actualmente.
- **Implementación de una Plataforma en la Nube Privada para servicios de IoT:** En este capítulo se desarrolla la solución llevada a cabo, detallando la arquitectura del sistema y los elementos que lo componen y explicando las decisiones tomadas y pasos a seguir para la puesta en marcha de esta plataforma en la nube privada.
- **Validación de la Plataforma en la Nube Privada para servicios de IoT:** Muestra el proceso de validación del sistema global para comprobar su correcto funcionamiento, desde la captación de los datos por parte del sensor hasta la visualización de estos por parte de los usuarios a través de la aplicación web creada.
- **Conclusiones y líneas de trabajo futuras:** Por último, tras la implementación, validación y comparación, se hace un repaso de los resultados y conclusiones extraídas, proponiendo además, una serie de posibles mejoras o futuras líneas de investigación.

1.3. Distribución de tareas

A continuación, se muestra una lista de las tareas llevadas a cabo para la realización de este proyecto, así como la distribución temporal de las mismas:

1. Búsqueda de información
   - Conocimiento de las bases teóricas y tecnologías existentes para el desarrollo del proyecto.
   - Documentación y estudio de las opciones de servidor para una elección óptima del mismo.
2. Implementación del servidor web

- Creación de una máquina virtual con sistema operativo Linux e implementación y configuración del servidor Nginx.

3. Contenido web y almacenamiento de datos
   - Diseñar la estructura tanto de la base de datos como de la estructura de la interfaz de usuario.
   - Elección del lenguaje de programación y desarrollo de la página web.
4. Implementación del protocolo MQTT
   - Conexión del sensor a la Raspberry.
   - Instalación de Mosquitto en los clientes y creación de los *scripts* necesarios.
5. Validar el funcionamiento del sistema IoT desarrollado
6. Redacción de la memoria

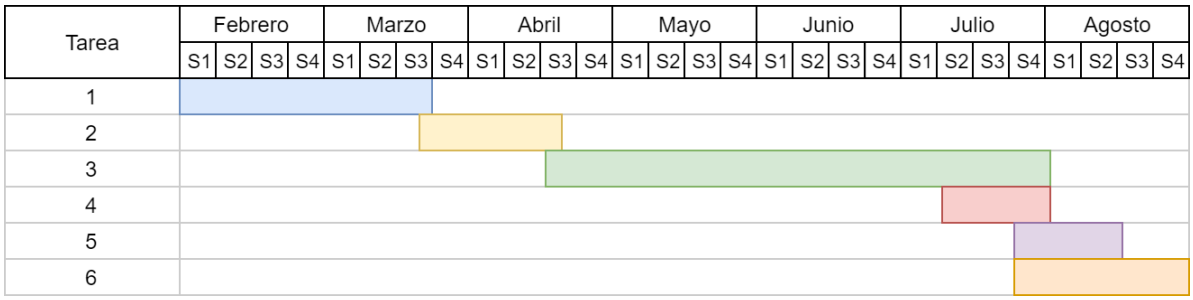


Figura 1. Distribución de tareas

2. Estado del arte

2.1. Internet de las cosas

El término IoT (Internet of Things, en inglés) hace referencia a la interconexión e integración del mundo físico con el mundo de la información, es decir, la red. Esto básicamente se traduce en la conexión de dispositivos y objetos a Internet y/o entre ellos mismos.

Este concepto está cada vez más presente en nuestra vida cotidiana. La razón de ello es que cuenta con un gran potencial, pues su objetivo es el desarrollo de aplicaciones y servicios para mejorar la calidad de vida de la sociedad, así como la automatización de tareas mejorando los procesos de ejecución, tiempos de producción y costes.

Como bien se ha mencionado, Internet of Things abre puertas a muchas y nuevas oportunidades de desarrollo, pero al existir un gran número de dispositivos conectados, también se plantean algunos problemas o desafíos. Entre ellos cabe destacar el problema de la seguridad, ya que se ha de garantizar la privacidad y el intercambio seguro de datos. Por otro lado, la gran cantidad de datos generada por dichos dispositivos es otro de los grandes desafíos a los que se enfrentan muchas empresas, puesto que han de ser capaces de reunir, almacenar, analizar e interpretar todos estos datos, es decir, se requiere una gestión eficiente de los mismos.

Las áreas en las que se puede implementar servicios IoT son muy diversas, entre ellas podemos citar aplicaciones de IoT para el sector industrial (hostelería, comercios, flotas de vehículos para logística, gestión de almacenes de logística, agricultura y ganadería…), aplicaciones de IoT para el uso doméstico (domótica, jardinería, salud…), aplicaciones de IoT en ciudades inteligentes (gestión de suministros, gestión ambiental, gestión del tráfico…). Gracias a la gestión eficiente y segura de los datos generados por estas aplicaciones se podrán definir nuevos modelos de mercado tanto horizontales como verticales en ámbitos como la educación, las comunicaciones, la ciencia, la política, el medioambiente…

2.1.1. Capacidades de un sistema IoT

Las capacidades que presenta un sistema IoT [1] se corresponden con los elementos básicos que lo conforman, como se muestra en la Figura 2.

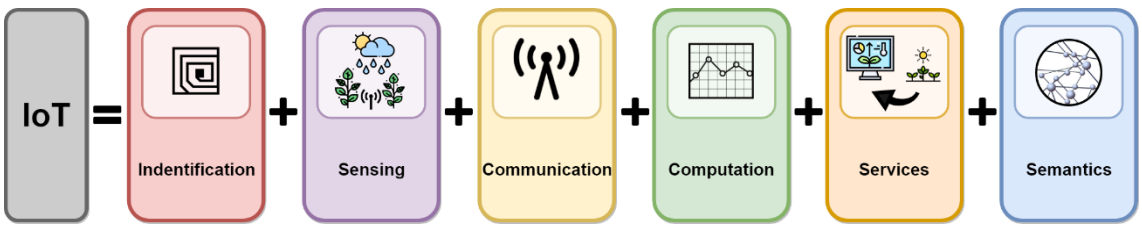


Figura 2. Capacidades de un sistema IoT

La **identificación** es la primera de las capacidades con las que debe contar un sistema IoT. Dentro de este bloque, por un lado, se encuentran los sistemas de identificación que ofrecen una ID única a cada elemento dentro de la red, es decir, un nombre específico para cada objeto. Algunos de estos métodos de identificación para IoT son: uCode (*uCode system*, en inglés) o EPC (*Electronic Product Code*, en inglés). Por otro lado, se debe considerar una dirección para cada uno de dichos objetos dentro de la red comunicaciones, donde se hace uso de los protocolos IPv6 e IPv4, aunque existen otras tecnologías alternativas como 6LoWPAN (adecuada para redes inalámbricas con una baja potencia). Cabe destacar que los métodos de identificación no son únicos de manera

global, por lo que sería la ruta de direccionamiento el modo de identificar inequívocamente estos objetos.

Captación: se trata de la recopilación de la información deseada de los objetos que se encuentran en la red. Esta información es recogida por sensores o actuadores normalmente conectados a placas hardware de bajo coste como pueden ser: Raspberry Pi, Arduino...

Comunicación: el objetivo es la conexión inalámbrica entre los nodos que conforman el sistema. Existen varias tecnologías que se usan comúnmente para los sistemas de IoT, en función del alcance y las prestaciones que ofrece cada una de ellas, podemos encontrar: tecnologías móviles 3GPP de cuarta (4G) y quinta generación (5G), tales como, NB-IoT, eMTC; Lora/LoRaWAN; WiFi; Bluetooth…

Computación: hace referencia al procesamiento de los datos, el cual puede realizarse desde diferentes tipos de plataformas. Por un lado, encontramos las plataformas propietarias y por otro las plataformas en la nube (*Cloud Computing*), las cuales permiten ofrecer servicios a los usuarios sin requerir un conocimiento especifico de estos. Este último tipo se desarollará con mayor detalle más adelante en este mismo documento.

Servicios: en este bloque se gestionan los datos brindando diferentes servicios a los usuarios, tales como:

- **Servicios de identidad vinculada**, en los que se identifican y vinculan objetos entre el mundo real y el virtual. Este es el tipo más sencillo de servicio que existe, y también el que más importancia tiene.
- **Servicios de agrupación de información**: hacen referencia a los que llevan a cabo la recopilación y asociación de los datos que se necesitan para las aplicaciones IoT.
- **Servicios de cooperación**: estos servicios colaboran con los servicios anteriores haciendo uso de sus datos para la toma de decisiones consecuentes.
- **Servicios ubicuos**: proporcionan servicios de cooperación independientemente del momento y lugar.

Por último, se encuentra el bloque de la **semántica**, teniendo tanta importancia como el resto de las capacidades, puesto que es aquí donde se ofrecen los servicios de la aplicación del usuario final gracias a los datos que han sido procesados y analizados.

2.1.2. Arquitectura de un sistema IoT

En cuanto a la arquitectura de IoT, actualmente no existe un acuerdo sobre el modelo a utilizar, por ello, diferentes instituciones y/o investigadores han propuesto modelos equivalentes [2]. De entre todos ellos, básicamente se pueden agrupar en los 3 tipos que se muestran a continuación:

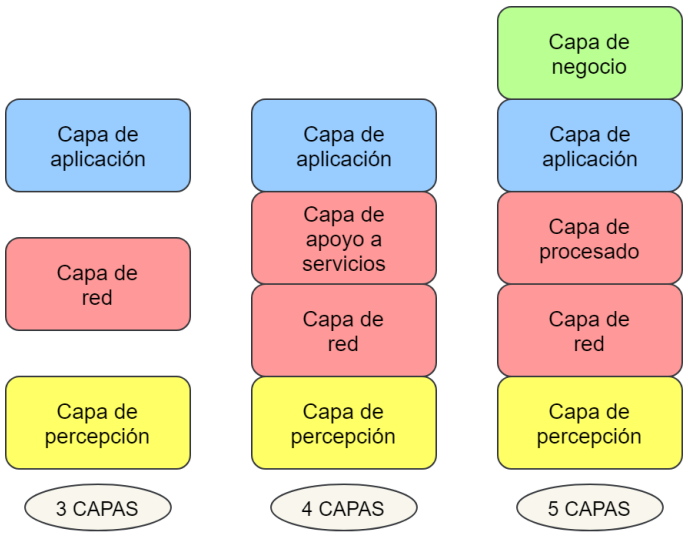


Figura 3. Tipos de Arquitectura IoT

- **Arquitectura de 3 capas**

Comenzando por la arquitectura de 3 capas, se puede observar a simple vista que se trata de la estructura más básica de las propuestas. Como su nombre indica, está formada por 3 capas, las cuales se explican a continuación:

- **Capa de percepción:** Hace referencia a la capa física. En esta capa es donde actúan los sensores, captando la información requerida para cada tipo de aplicación, como pueden ser diferentes parámetros físicos o la identidad de otros objetos inteligentes del entorno.
- **Capa de red:** Esta capa es la que se encarga de la conexión entre objetos inteligentes, o de la conexión de estos con servidores y dispositivos de red. En esta arquitectura de 3 capas, esta también se encarga de transmitir y procesar los datos recogidos por los sensores.
- **Capa de aplicación:** Esta capa es la responsable de dar los servicios de cada tipo de aplicación a los usuarios. Entre los tipos de aplicaciones en las que se puede implementar IoT podemos encontrar hogares, ciudades inteligentes, …

Esta arquitectura permite entender fácilmente la idea principal que supone IoT. No obstante, aunque se puede hacer uso de este modelo en aplicaciones sencillas que no requieren una ampliación de las capacidades de red, no es suficiente para llevar a cabo investigaciones sobre Internet de las cosas. Debido a ello encontramos otras propuestas que incorporan capas adicionales y/o desglosan en varias capas las funciones que en el anterior modelo eran desempeñadas por una sola.

- **Arquitectura de 4 capas**

La arquitectura de cuatro capas es bastante similar a la anteriormente mencionada. Su principal diferencia con la de cinco, es que no cuenta con la denominada capa de negocio, en la cual se gestiona todo el sistema IoT, incluyendo tanto las aplicaciones como los modelos de negocio, ganancias y privacidad de los usuarios.

A continuación, se desarrolla el modelo de cinco capas, ya que generalmente es el más utilizado:

- **Arquitectura de 5 capas**

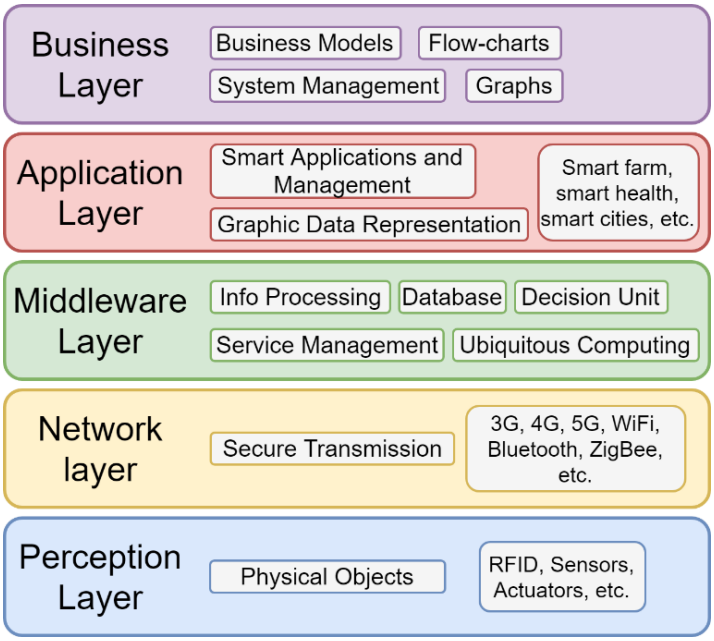


Figura 4. Arquitectura IoT de cinco capas

- **Capa de percepción o de objetos:** Hace referencia a la capa física. En esta capa es donde actúan los sensores, captando la información requerida para cada tipo de aplicación, como pueden ser diferentes parámetros físicos o la identidad de otros objetos inteligentes del entorno.
- **Capa de red o de abstracción:** En estas arquitecturas, la capa de transporte se encarga de transferir los datos recogidos por los sensores desde la capa de percepción hasta la capa de procesado y viceversa través de redes (NB-IoT, LoRa, LAN, Bluetooth, RFID, NFC, entre otros).
- **Capa de procesado (middleware layer):** En esta capa se almacenan y analizan los datos que llegan de la capa de transporte. Puede gestionar y proporcionar diferentes servicios a las capas inferiores. En esta capa se emplean tecnologías como bases de datos, *Cloud Computing* y módulos de procesamiento de *Big Data*.
- **Capa de aplicación:** Esta capa es la responsable de dar los servicios de cada tipo de aplicación a los usuarios. Como se ha mencionado antes, entre los tipos de aplicaciones en las que se puede implementar IoT podemos encontrar hogares, agricultura, ciudades inteligentes, …
- **Capa de negocio:** Esta capa gestiona todo el sistema IoT, incluyendo aplicaciones, modelos de negocio, ganancias y privacidad de los usuarios. Al gestionarse las actividades, los servicios y los resultados, esta capa ofrece la posibilidad de llevar a cabo optimizaciones y mejoras en el sistema IoT. Por ello la mayoría de los fabricantes hacen uso de la arquitectura que incluye esta capa.

2.1.3. Cloud Computing para servicios de IoT

La computación en la nube o *Cloud Computing* es una tecnología que permite ofrecer servicios mediante la conexión a Internet, basándose en servidores accesibles desde cualquier punto y en cualquier momento.

La computación en la nube presenta características interesantes para el concepto de Internet de las Cosas ya que permite el acceso ubicuo, almacenamiento y gestión de los datos recogidos por los dispositivos, ofreciendo escalabilidad y elasticidad. Esto supone una gran ventaja a la hora de desarrollar aplicaciones de IoT en diferentes ámbitos.

2.1.3.1. Arquitectura de Cloud Computing

Respecto a la arquitectura de Cloud Computing, esta se puede dividir en dos secciones conectadas entre sí mediante Internet, tal y como se aprecia en la Figura 5.

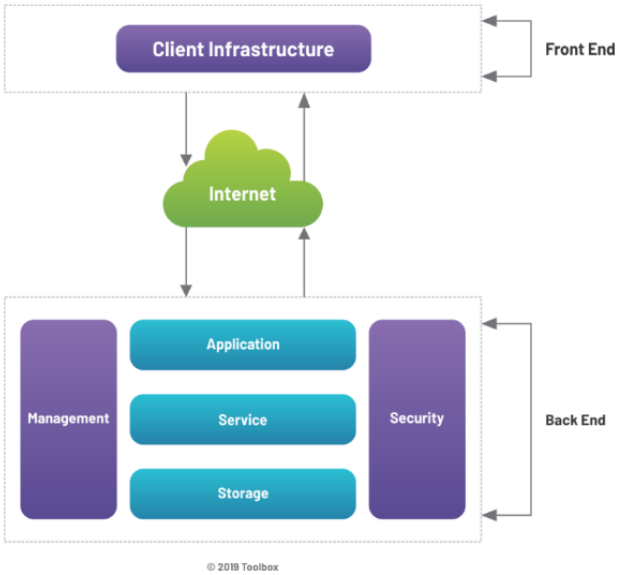


Figura 5. Arquitectura de Cloud Computing [3]

Arquitectura *front-end*: consiste en la parte de la arquitectura orientada al usuario final en la que se incluye:

- **Software** e **Interfaz de usuario**, se ejecuta del lado del cliente y es la parte visible para el usuario, donde este interactúa.
- **Dispositivo del cliente**, el cual no requiere de gran potencia informática, puesto que la mayoría del procesamiento se realiza en la nube.

Arquitectura *back-end*: esta arquitectura es responsabilidad del proveedor del servicio en la nube. Impulsa la arquitectura *front-end*. Se compone de varios elementos:

- La **aplicación** que se pone a disposición del usuario final. Aquí se coordinan las necesidades del cliente con los recursos en la parte *back-end*.
- **Servicio**, se trata del desempeño de tareas de computación en la nube. Algunos ejemplos de servicios pueden ser: almacenamiento, servicios web o entornos de desarrollo de aplicaciones.
- **Almacenamiento** de los datos necesarios para la ejecución del software en la nube.

- Servicios de **administración** de los recursos de forma eficiente para conseguir un funcionamiento del sistema fluido.
- **Seguridad**, para proteger al servidor de pérdidas de datos o ataques. También se realizan copias de seguridad del almacenamiento para garantizar el correcto funcionamiento del sistema.

2.1.3.2. Modelos de Servicio de Cloud Computing

En cuanto a los modelos de servicio [4] que ofrece la computación en la nube, encontramos tres posibilidades:

- **SaaS (*Software as a Service*):** En este modelo de servicio la capacidad que posee el usuario es la hacer uso de los recursos del proveedor de SaaS, pudiendo ser estos aplicaciones o software que se ejecutan en una infraestructura de la nube (por ejemplo, el correo electrónico web). Por tanto, como se refleja en el esquema de la Figura 6, el usuario no dispone de un control sobre ninguno de los elementos de la infraestructura (red, almacenamiento, servidores, sistemas operativos...), exceptuando que se le permita algún tipo de configuración o personalización.

  Algunos ejemplos de este modelo son: Google Docs, Gmail, Dropbox, …

- **PaaS (*Platform as a Service*):** El usuario puede desplegar sus propias aplicaciones sobre la plataforma del proveedor, el cual suele ofrecer también herramientas de programación para facilitar el desarrollo de dichas aplicaciones. Debido a ello, en este modelo de servicio el usuario puede controlar la aplicación, sin embargo, no puede administrar la infraestructura subyacente de la nube como se indica en la Figura 6.

  Ejemplos de *Platform as a Service* son: Google App Engine, Openshift, Cloud Foundry, ...

- **IaaS (*Infraestructure as a Service*):** En este modelo de servicio, la capacidad del usuario aumenta, incluyendo desde el control de las aplicaciones desarrolladas hasta los sistemas operativos (Figura 6). Sin embargo, es el proveedor el que se encarga de la comunicación, el procesamiento o el almacenamiento.

  Entre los ejemplos de este modelo de servicio encontramos: Amazon Web Services, Google Compute Engine, Microsoft Azure, ...

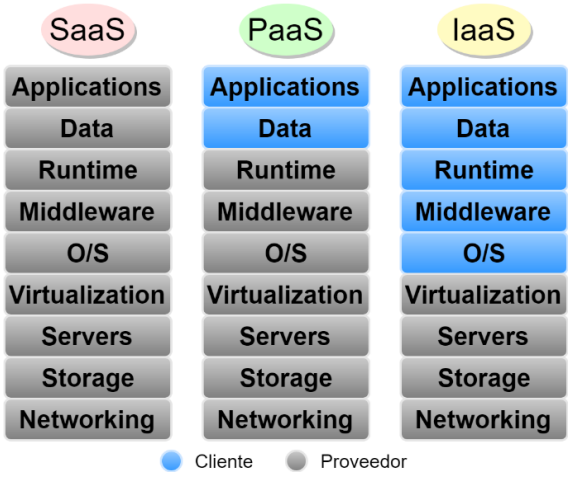


Figura 6. Modelos de servicio de Cloud Computing

2.1.3.3. Modelos de implementación de Cloud Computing

Principalmente hay cuatro modelos de implementación de computación en la nube, los cuales son presentados a continuación [5]:

1. **Nube Privada:** La infraestructura en la nube se proporciona para uso exclusivo de una organización. Puede ser propiedad de dicha organización y ser administrada por ella, por un tercero o una combinación de ambos. Comúnmente es implementada dentro de las instalaciones de la misma empresa, pero también puede encontrarse en una ubicación externa a ella.
2. **Nube Comunitaria:** Esta infraestructura está dedicada a una comunidad de organizaciones con inquietudes similares (misiones o metas, políticas de conformidad, requisitos de seguridad…). Al igual que en el caso de una nube privada esta puede encontrarse dentro o fuera de las instalaciones de alguna de las organizaciones integrantes, así como ser gestionada por una o varias de ellas o incluso un tercero.
3. **Nube Pública:** Esta infraestructura está prevista para el libre acceso del público en general, presentando posibles restricciones. Esta puede ser propiedad y estar administrada por una o varias organizaciones empresariales, académicas o gubernamentales. Se aloja en las instalaciones del proveedor de la nube.
4. **Nube Híbrida:** Este modelo se compone de dos o más nubes distintas de las descritas anteriormente que siguen siendo entidades únicas, pero se encuentran vinculadas bajo una misma tecnología estandarizada o propietaria que permite la portabilidad de datos y aplicaciones. Un ejemplo de esto sería la ráfaga de nubes para equilibrar la carga entre las nubes que lo componen.

2.1.3.4. Plataformas de Cloud Computing para IoT

Debido a la necesidad de gestionar de forma ágil y eficiente el gran volumen de datos que generan los diferentes sistemas IoT surgen las plataformas de computación en la nube que ofrecen servicios orientados a Internet de las Cosas. En los siguientes subapartados se presentan algunas de estas plataformas explicando también sus puntos más característicos.

2.1.3.4.1. Plataformas de Cloud Computing Open Source

Entre las plataformas de Cloud Computing Open Source para IoT que se pueden encontrar se ha hecho una selección de tres ejemplos significativos [6]. A continuación, se presentan cada uno de ellos, exponiendo sus características principales:

- **Kaa IoT Platform**

Kaa es una tecnología *middleware* de código abierto aplicable al desarrollo de IoT empresarial a cualquier escala [7]. Esta plataforma ofrece soluciones de IoT en numerosos ámbitos, entre los que destacan: agricultura, ciudades inteligentes, comercio, deporte, telecomunicaciones, salud…

Entre los puntos fundamentales que caracterizan a Kaa IoT Platform se encuentran:

- Cuenta con una **arquitectura de microservicios**, ofreciendo así un alto grado de personalización, ya que estos se pueden reorganizar e integrar con sistemas de terceros.

- Es **independiente de la tecnología**, lo que da libertad a los desarrolladores para implementar aplicaciones prácticamente con cualquier lenguaje de programación.
- Además, ofrece también **libertad de despliegue** pudiendo implementar los servicios de Kaa en centros de datos, máquinas físicas o virtuales, infraestructuras de nube pública, híbrida o privada. Esta última permite contar con el control total del sistema, aumentando así la seguridad.
- Admite el uso de **protocolos IoT abiertos** como son MQTT y CoAP. MQTT es el protocolo utilizado por Kaa de forma predeterminada.
- Se trata de una plataforma **escalable**, **elástica** y **auto-reparable**, pudiendo admitir tantos clientes como se necesiten y restaurando la plataforma ante posibles fallos.
- En cuanto a la **seguridad**, TLS o DTLS son los certificados usados por defecto en las comunicaciones de Kaa con los dispositivos. Así mismo, Kaa ofrece una gestión flexible en lo que al ciclo de vida de las credenciales se refiere.
- Ofrece **soporte de puerta de enlace**, lo que significa que se puede optar por conectar los dispositivos directamente de forma individual o mediante una conexión multiplexada donde varios dispositivos utilizan la misma conexión al servidor. Esta última opción puede ser interesante por ejemplo para dispositivos con poca potencia como LoRaWAN o ZigBee.
- Presta **servicios de los proveedores,** ofreciendo asistencia profesional y soporte de producción.

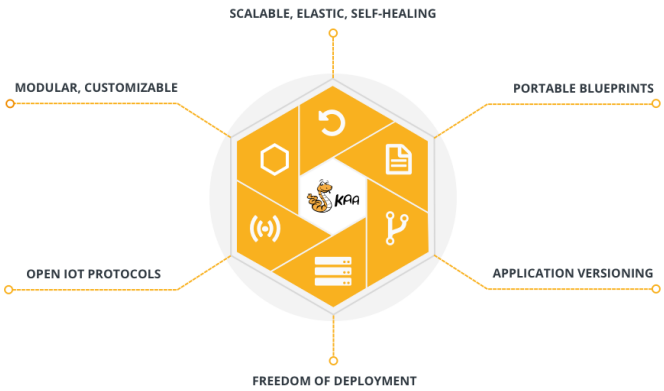


Figura 7. Características principales de Kaa IoT

- **OpenIoT**

OpenIoT es una plataforma de *middleware* de código abierto que unifica IoT con los servicios de computación en la nube. Su infraestructura permite una configuración flexible y la creación de algoritmos con los que recoger y filtrar los diferentes flujos de datos procedentes de los objetos conectados a Internet [8]. La Figura 8 es una representación simple del funcionamiento de esta plataforma.

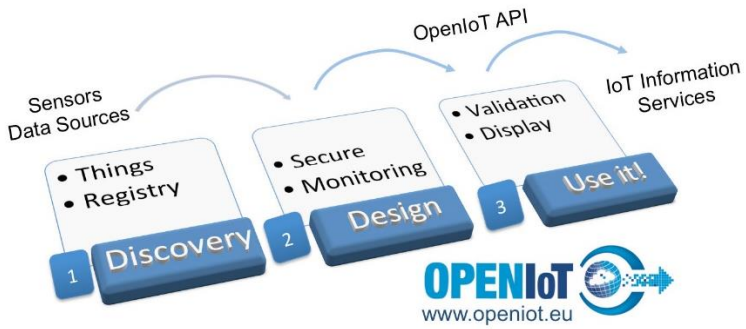


Figura 8. Diagrama de funcionamiento de OpenIoT

Las capacidades más importantes de OpenIoT:

- Ofrece el registro, implementación y descubrimiento de sensores según su ubicación y tipo.
- Capacidad de monitorizar el estado de los servicios de IoT, los datos que se manejan en cada uno y el estado de los sensores correspondientes.
- Asegura la interoperabilidad semántica de los servicios IoT y los flujos de datos en la nube.
- Presenta autenticación y autorización para el acceso a estos recursos.
- Permite la visualización de la información correspondiente a los servicios IoT en diferentes formatos como mapas, gráficos de diferentes tipos o paneles.
- Cuenta con la posibilidad de desarrollar aplicaciones sencillas prácticamente sin programar, lo que acelera el proceso de creación de dichas aplicaciones.

- **ThingSpeak**

ThingSpeak es una plataforma de análisis de IoT de código abierto que permite la recogida de los datos captados por los sensores, su almacenamiento en la nube y su posterior visualización y análisis [8]. Además, posibilita la creación de prototipos y construcción de sistemas IoT sin tener que configurar servidores o desarrollar software web. La Figura 9 muestra un diagrama de lo que sería su estructura básica.

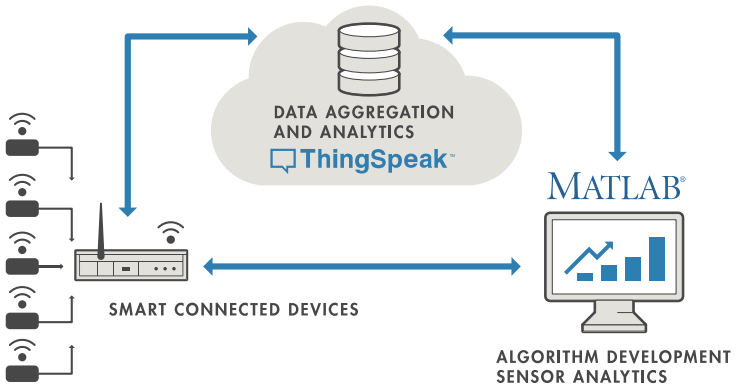


Figura 9. Estructura básica de ThingSpeak

Las principales capacidades que presenta esta plataforma se pueden resumir en los siguientes puntos:

- Presenta una configuración sencilla de los dispositivos que efectúan la **recogida de los datos** y envío de estos a la nube haciendo uso de protocolos de IoT populares. ThingSpeak almacena por defecto los datos en canales privados, pero también ofrece canales públicos donde los datos pueden ser compartidos.
- Ofrece el **análisis** de los datos almacenados. La característica que diferencia a esta plataforma de otras es que proporciona acceso a MATLAB®. Esto permite visualizar los datos en gráficos, diagramas o medidores y facilita la obtención de algoritmos, patrones o modelos predictivos para los sistemas IoT.
- Permite establecer **respuestas automáticas** según el resultado del análisis de dichos datos, pudiendo crear alertas y comunicarse a través de servicios de terceros como Twilio® o Twitter®.

2.1.3.4.2. Plataformas de Cloud Computing No Open Source

Las principales plataformas de Cloud Computing que no son de código abierto para IoT que podemos encontrar hoy en día en el mercado son: AWS IoT, Azure IoT y Google Cloud IoT.

- **Amazon Web Services IoT (AWS IoT)**

AWS IoT es un servicio cuya finalidad es poder recopilar, almacenar y analizar datos recogidos de los dispositivos conectados a Internet, pudiendo crear también aplicaciones destinadas a los usuarios con el objetivo de que puedan controlar estos dispositivos. Para conseguir esto, AWS se encarga de establecer una comunicación bidireccional entre sensores, actuadores o aparatos inteligentes y la nube de AWS [10].

Algunos de los elementos que caracterizan a AWS IoT se describen a continuación:

- **Gateway de dispositivos**, permitiendo una comunicación segura y eficaz con AWS IoT.
- Cuenta con un **agente de mensajes**, el cual utiliza MQTT o MQTT sobre WebSocket para la publicación y recepción entre los dispositivos y aplicaciones AWS IoT.
- Hace uso de un **motor de reglas** que permite procesar mensajes y enviar los datos a otros servicios de AWS (Amazon DynamoDB, AWS S3, ...)
- Comprende servicios de **seguridad e identidad**, protegiendo el envío de datos desde los dispositivos con credenciales.
- Proporciona el **registro** de los dispositivos para facilitar la organización de los recursos asociados a estos en la nube de AWS.
- Cuenta con un servicio denominado **Device Shadow** con el que se publica información del estado de los dispositivos para que pueda ser usada por aplicaciones u otros dispositivos conectados.
- Permite definir una estrategia de **autenticación personalizada**.
- Ofrece un **Servicio de Jobs**, con el cual se puede especificar un conjunto de operaciones remotas a uno o más dispositivos para que sean ejecutadas.

La Figura 10 representa un diagrama del funcionamiento de AWS IoT.

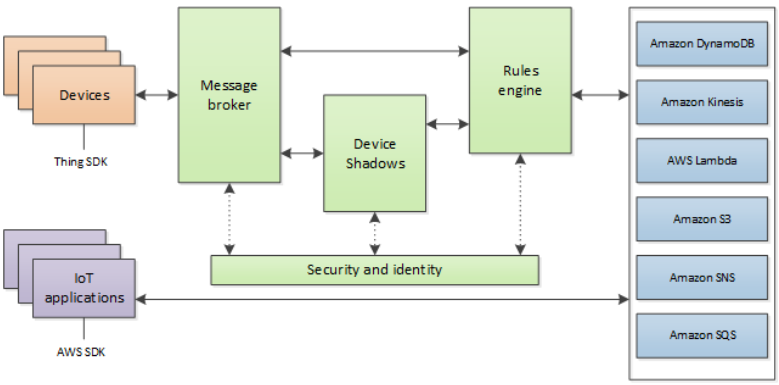


Figura 10. Diagrama de flujo de AWS IoT

AWS es integrado directamente con otros servicios de AWS como son: Amazon S3 (almacenamiento escalable en la nube de AWS), Amazon DynamoDB (bases de datos NoSQL) y AWS Lambda (ejecución de código en servidores virtuales de Amazon EC2) entre otros.

- **Azure IoT**

Esta plataforma creada por Microsoft ofrece a los usuarios la capacidad de crear soluciones IoT según sus necesidades, contado para ello con varias tecnologías y soluciones de IoT propias de Azure. La Figura 11 muestra un resumen y clasificación de estas tecnologías y soluciones.

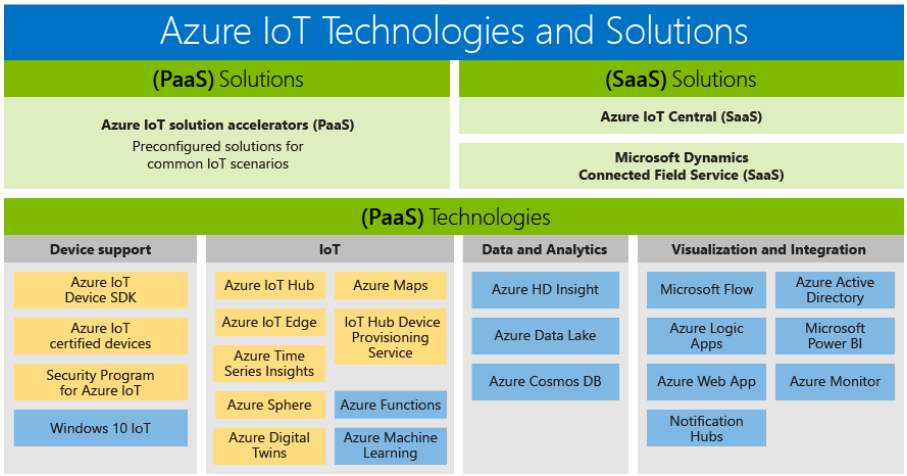


Figura 11. Tecnologías y Soluciones de Azure IoT [11]

Como se observa, Azure IoT cuenta con diversas tecnologías para ofrecer soluciones de IoT, esto supone una gran flexibilidad para que el usuario pueda adaptarlo a sus necesidades. Por consecuencia, no existe una arquitectura fija para describir una aplicación IoT que haga uso de esta plataforma, sin embargo, Azure IoT muestra como arquitectura de referencia el esquema representado en la Figura 12.

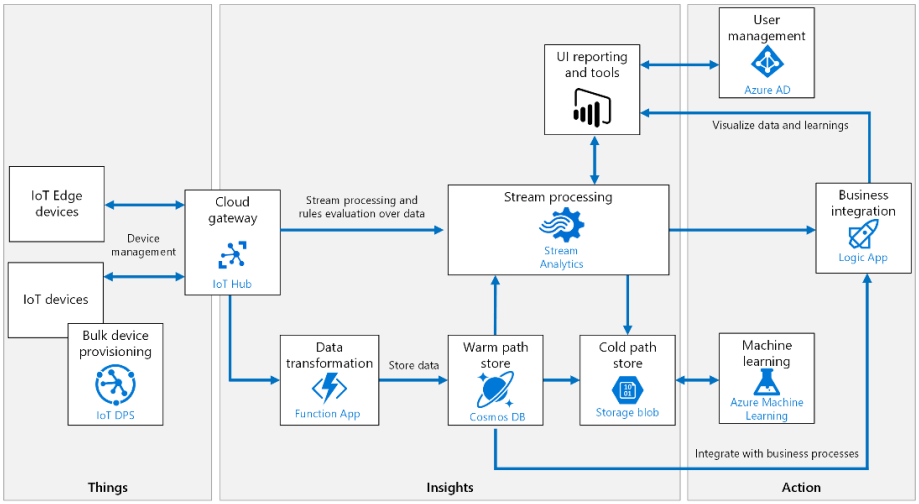


Figura 12. Arquitectura propuesta para un sistema Azure IoT [12]

Esta arquitectura propuesta hace uso de componentes PaaS (plataforma como servicio) de Azure. A continuación, se presentan los puntos más importantes que la componen:

- ***IoT devices***, hace referencia a los dispositivos IoT que se registran y se conectan a la nube para el envío y recepción de datos. ***IoT Edge devices*** corresponde a dispositivos que son capaces de procesar datos en el mismo dispositivo, lo que se consigue gracias a *Azure IoT Edge*.
- Seguidamente se encuentra el bloque de ***Cloud Gateway,*** el cual consiste en una puerta de enlace en la nube para la conexión y envío de datos de los dispositivos, incluyendo también la capacidad de administración y control de estos. Esta tarea se realiza gracias a la tecnología *Azure IoT Hub*.
- ***Device provisioning***, desempeñado por *IoT Hub Device Provisioning Service (IoT DPS)*, permite el registro de dispositivos, conectar grupos de dispositivos y asignarlos a un punto final de Azure IoT.
- ***Stream processing*** se trata del análisis y procesado de grandes flujos de datos. Una de las herramientas que desarrollan esta tarea es *Azure Stream Analytics*. Por otro lado, *Azure Machine Learning* permite el aprendizaje automático tras el análisis de estos datos, consiguiendo así algoritmos predictivos.
- En Azure IoT el almacenamiento de datos se puede dividir por un lado en ***Warm path storage*** gracias a ***Azure Cosmos DB***, donde se almacenan datos que necesitan estar disponibles de forma instantánea para informes y visualización y por otro lado ***Cold path storage*** por *Azure Blob Storage*, donde se almacenan datos a largo plazo para su procesado en bloques.
- El bloque denominado ***Data transformation*** consiste en el tratamiento de los datos y su transformación si fuese necesaria. La tecnología *Azure Functions* se encarga de realizar este proceso.
- ***Business process integration***, donde se integran procesos empresariales gracias a *Azure Logic Apps*. Esto permite llevar a cabo acciones de forma automática en función de la información que ofrecen los datos de los dispositivos.
- ***User management***, permite gestionar las acciones que pueden realizar los usuarios en los dispositivos, gracias a la autenticación y autorización por *Azure Active Directory*.

- **Google Cloud IoT**

Esta plataforma está compuesta por un conjunto de herramientas que hacen posible tanto la conexión de dispositivos como el procesado, almacenamiento y análisis de datos en la nube. Google Cloud IoT admite una gran variedad de sistemas operativos, destacando especialmente su funcionamiento con Debian Linux.

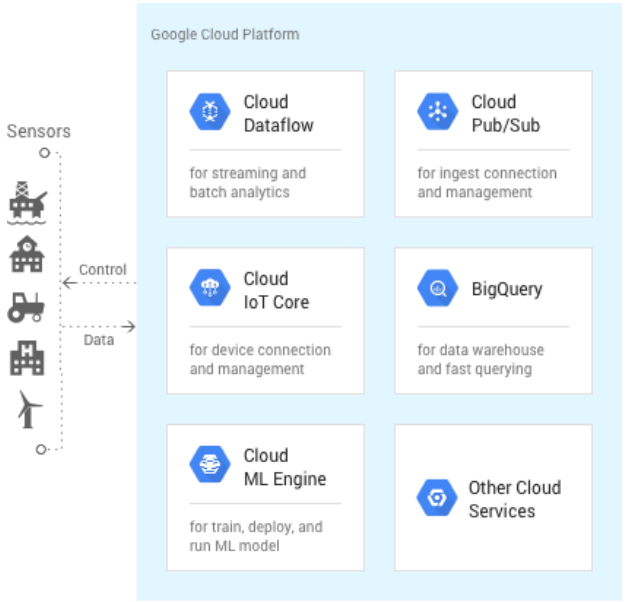


Figura 13. Arquitectura de referencia de Google Cloud IoT [13]

Google Cloud IoT sugiere la arquitectura de referencia mostrada en la Figura 13, donde se distinguen algunas de las herramientas que Google Cloud propone para soluciones de IoT. Seguidamente se expone el propósito principal de cada una de ellas:

- *Cloud IoT Core* se trata del servicio principal de esta plataforma, su objetivo es la conexión, gestión e ingestión de datos de diferentes dispositivos, ofreciendo seguridad y sencillez en el proceso. Si se combina con otros servicios de Cloud Cloud IoT se consigue una solución completa abarcando la recogida, procesado, análisis y visualización de datos IoT.
- *Cloud Pub/Sub* es un servicio de mensajería que permite el envío y recepción de mensajes entre aplicaciones independientes, así como la distribución de datos entre proyectos y aplicaciones en entornos híbridos. Presenta una latencia baja y escalabilidad bajo demanda.
- La herramienta *BigQuery* se caracteriza por ser un almacén de datos sin servidor que hace uso del lenguaje SQL y es altamente escalable según las necesidades de almacenamiento y la potencia informática.
- *Cloud Dataflow* se encarga del procesamiento de datos en tiempo real (*streaming*) y por lotes. Se caracteriza por su fiabilidad y presentar una alta capacidad de procesamiento ya que no necesita de un servidor.
- *Cloud Machine Learning Engine* (*Cloud ML Engine*) ofrece servicios de preparación y predicción en función del análisis de datos que se ha realizado. Esto permite el desarrollo de modelos de aprendizaje automático y su uso en entornos de producción.

3. Implementación de una Plataforma en la Nube Privada para servicios de IoT

Este apartado desarrolla la implementación práctica de una solución privada de plataforma en la nube para servicios IoT. Para ello, primero se muestra un esquema de la arquitectura global del sistema, citando los elementos que la componen. Seguidamente se hace hincapié en cada uno de estos componentes, describiendo tanto el montaje realizado para cada una de las partes y las instalaciones de software necesarias como la justificación de las tecnologías y/o lenguajes de programación de los que se hacen uso.

3.1. Arquitectura del sistema

La Figura 14 representa la arquitectura del sistema IoT implementado, el cual seguiría el modelo de cuatro capas. En ella se pueden distinguir los diferentes elementos que lo componen: nodo cliente, broker MQTT, nodo servidor y usuarios finales.

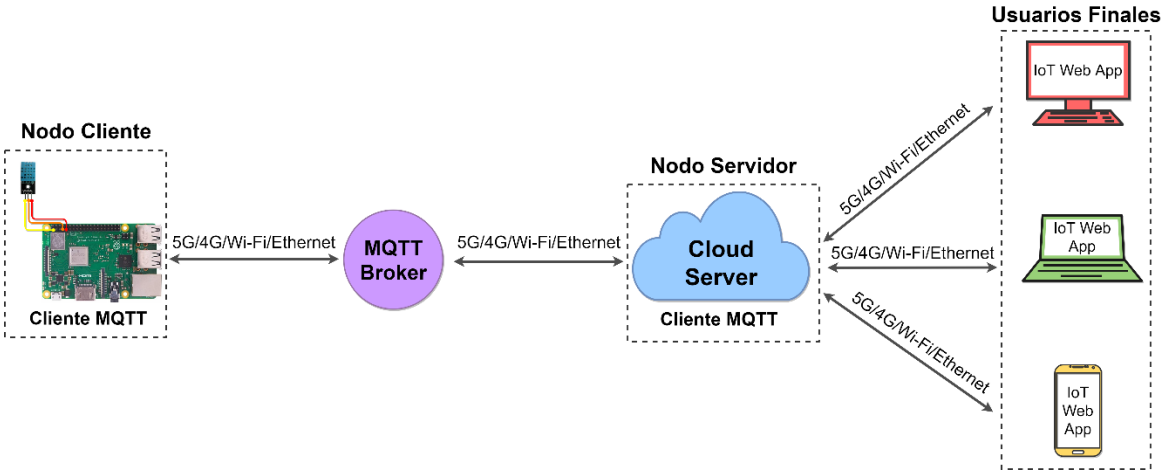


Figura 14. Arquitectura del sistema IoT implementado

3.1.1. Sistema del nodo cliente

Como se ha indicado anteriormente, la primera capa de la arquitectura de un sistema IoT es la de percepción, la cual permite obtener los datos del mundo físico o virtual mediante sensores gestionados por placas de desarrollo.

Para implementar este nodo, se ha elegido una **Raspberry Pi 3 B+**, a la cual se le ha instalado previamente Raspbian como sistema operativo y se le ha conectado un **sensor** para obtener datos reales del medio ambiente, con el fin de comprobar el correcto funcionamiento del sistema IoT implementado.

El sensor escogido es el DHT11[14], el cual proporciona datos de temperatura y humedad del entorno en el que se coloque. Al tratarse de un sensor *ultra low cost* no presenta una gran precisión en sus medidas pues no ofrece ningún decimal, pero es suficiente para comprobar el correcto desempeño del sistema implementado. Un resumen de sus características técnicas se contempla en el Anexo I de este documento.

Para la correcta conexión del sensor se ha consultado el manual de uso de la Raspberry Pi 3 B+, donde se ha podido identificar el esquema de pines de la placa, como se muestra en la Figura 15.

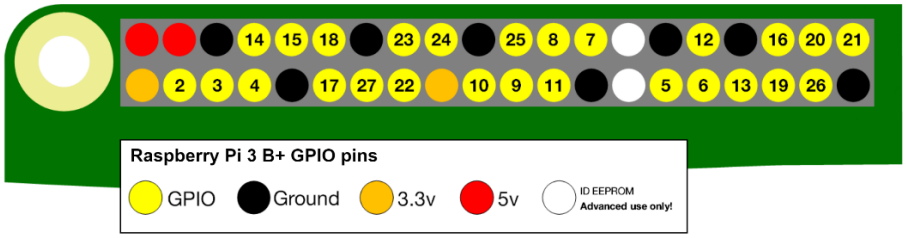


Figura 15. Esquema de pines de Raspberry Pi 3 B+ [15]

La Figura 16 representa la elección de pines que se ha hecho en este caso para la conexión entre el sensor y la Raspberry. El pin (+) del DHT11 corresponde a la señal de 3.3V por lo que ha de conectarse a uno de los pines de la Raspberry habilitados para ello (en este caso se usará el primer pin de la segunda fila). El pin (-) hace referencia a tierra, en este caso se ha escogido el quinto pin de la segunda fila para ello. Por último, el pin (*out*), corresponde a la señal de datos, este puede conectarse a cualquier pin GPIO (*General Purpose Input/Output*) de la Raspberry Pi 3 B+, eligiéndose en esta ocasión el GPIO4. Al tratarse del modelo de 3 pines no es necesaria la utilización de una resistencia en la conexión de este sensor a la Raspberry Pi 3 B+, ya que se encuentra integrada en la pequeña placa a la que va soldado el sensor y, por tanto, se puede hacer una conexión directa entre la salida (+) y el pin de 3.3V de la Raspberry.

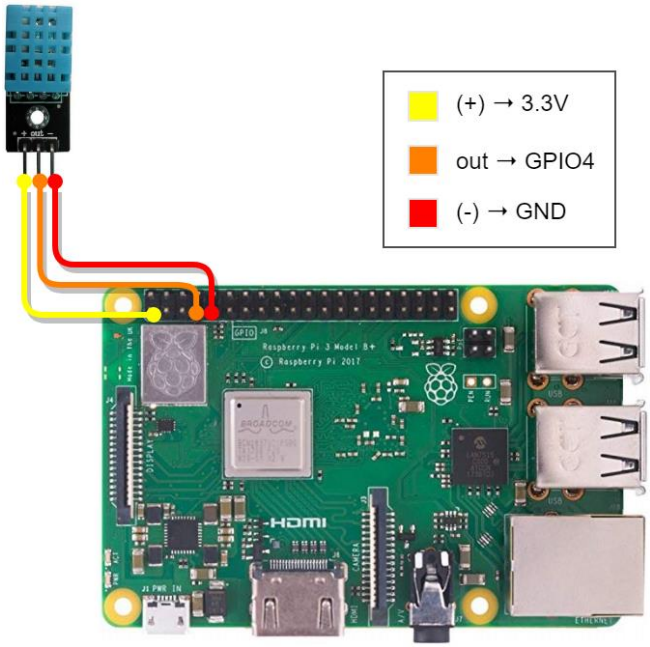


Figura 16. Conexión del sensor DHT11 a la Raspberry

Una vez se ha llevado a cabo la correcta conexión entre el sensor y la Raspberry, tal y como se muestra en la Figura 16, para poder realizar lecturas con el sensor se necesita de un código que indique los parámetros necesarios (como el tipo de sensor o el pin al que se encuentra conectado), inicialice el sensor y tome las medidas correspondientes. Para ello se ha optado por usar la librería de Adafruit[1], la cual se hace cargo de los datos necesarios a intercambiar con el sensor.

Esta librería escrita en Python acepta varios sensores, entre ellos el DHT11. Para poder realizar uso de ella se necesita escribir las instrucciones que se muestran a continuación en la Terminal de Raspberry:

[1] https://github.com/adafruit/Adafruit_Python_DHT

```
sudo apt-get update
sudo apt-get install build-essential python-dev (Instalación de paquetes
necesarios)

git clone https://github.com/adafruit/Adafruit_Python_DHT.git (Clonación de la
biblioteca Adafruit DHT de su repositorio)
cd Adafruit_Python_DHT

sudo python setup.py install
```

Tras introducir estos comandos, la biblioteca Adafruit DHT ya se encuentra instalada. Esto se puede comprobar ejecutando desde la misma Terminal de Raspberry el código Python de ejemplo que incluye la librería instalada, como respuesta se obtiene la medida de temperatura y humedad que realice el sensor en ese momento.

Además, puede ser utilizada en cualquier *script* Python. Esto es un dato importante, ya que como se verá más adelante esta biblioteca se tiene que importar dentro del código que se necesita para conseguir el envío de estos datos medidos por el sensor.

Una vez se cuenta con el hardware necesario listo para la recogida de datos, como el propósito final del sistema es que el usuario pueda acceder a estos datos de una manera fácil y cómoda desde la ubicación en la que se encuentre, se necesita de un protocolo de comunicación para enviar la información al nodo servidor donde se almacenará en la base de datos.

El protocolo que se ha elegido para la comunicación del cliente con el servidor es el protocolo de Transporte de Telemetría de Cola de Mensajes MQTT (*Message Queue Telemetry Transport*, en inglés), por ser uno de los protocolos más utilizados en la comunicación de dispositivos IoT. MQTT fue desarrollado en 1999 por Dr. Andy Standford-Clark de IBM, y Arlen Nipper de Eurotech como un mecanismo para conectar sensores en la industria petrolera. A partir del 2014, paso a ser un estándar de código abierto según la OASIS (*Advancing Open Standards for the Information Society*, en inglés). Este protocolo está basado en la publicación de mensajes y en la subscripción de los clientes a los temas en los que se publican esos mensajes. A los clientes que están esperando la llegada de mensajes únicamente reciben aquellos que se publican en los *topics* a los que se han subscrito. Está basado en la pila TCP/IP como base de la comunicación [16][17].

Dentro de las principales características de MQTT, podemos mencionar:

- Es un protocolo ligero. Posee un encabezado simple. Esto permite minimizar el ancho de banda, perfecto para aplicaciones de IoT en las que se envían cantidades pequeñas de información.
- Entrega los mensajes pendientes al conectarse los clientes.
- Permite mantener una comunicación bidireccional entre dispositivos.
- Debido a su estructura presenta un alto grado de escalabilidad. Además, cuenta con otras características como seguridad y fiabilidad.
- Requiere de poco consumo en los dispositivos que lo implementan.

La arquitectura de MQTT sigue una topología de estrella, contando con dos tipos de entidades (broker y cliente), tal y como se muestra en la Figura 17.

- Broker: funciona como servidor del sistema. Por el pasan todos los mensajes publicados, los cuales se encaminan a los clientes subscritos, en función de los servicios que tenga contratado cada cliente.

- Cliente: se trata de cualquier dispositivo que interactúe con el broker enviando y/o recibiendo mensajes.
- Los mensajes que publican los clientes se hacen en temas (*topics*).

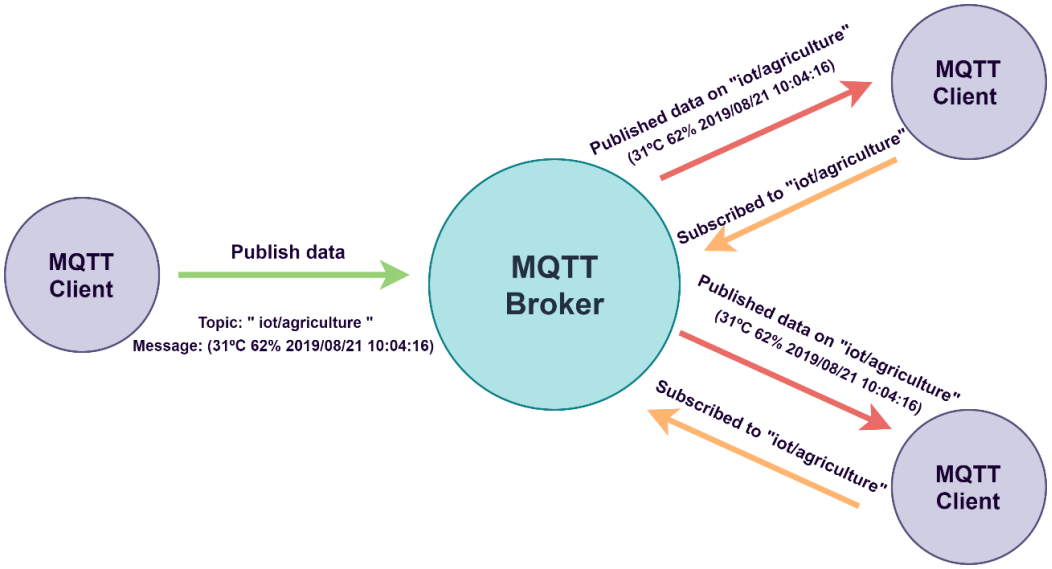


Figura 17. Esquema de funcionamiento MQTT

MQTT dispone de un mecanismo de calidad del servicio o QoS que permite gestionar la robustez del envío de mensajes al cliente ante fallos. En el protocolo se definen 3 calidades de servicio diferentes:

- QoS 0: El mensaje es transmitido como máximo una única vez sin esperar una confirmación de que se ha recibido. Este nivel es el usado por defecto.
- QoS 1: El cliente envía el mensaje hasta que el broker confirma su envío a la red, por lo que asegura que el mensaje se entrega al menos una vez.
- QoS 2: Este nivel de calidad garantiza que el mensaje se entrega estrictamente una vez.

El cliente que publica un mensaje tiene la capacidad de escoger el nivel de calidad con el que lo envía, no obstante, la calidad de servicio máxima estará delimitada por la que fije el cliente que recibe estos mensajes.

A medida que se incrementa nivel de QoS el sistema se vuelve más confiable (a la hora de que lleguen los mensajes correctamente) pero supone una mayor latencia y requerimiento de ancho de banda.

Para la implementación de los clientes se ha decidido hacer uso del mediador de mensajes de código abierto Mosquitto, ya que presenta una rápida y sencilla implementación y cuenta con un mayor abanico de compatibilidad con brokers MQTT. Los comandos necesarios para la instalación de un cliente Mosquitto son:

```
sudo wget http://repo.mosquitto.org/debian/mosquitto-repo.gpg.key
sudo apt-key add mosquitto-repo.gpg.key
cd /etc/apt/sources.list.d/
sudo wget http://repo.mosquitto.org/debian/mosquitto-stretch.list
sudo -i
apt-get update
apt-get install mosquito-clients
```

Para escribir el script correspondiente se requiere del uso de la librería de cliente en Python, Paho-mqtt, la cual implementa la versión 3.1 y 3.1.1 del protocolo MQTT. Además de una serie de paquetes para conseguir el almacenamiento en una base de datos MySQL (estos no son necesarios para el cliente que publica, sino para que el que se subscribe almacene la información, como se verá más adelante).

```
sudo apt update
sudo apt upgrade
sudo apt install python
sudo apt install python-pip
pip install paho-mqtt
pip install mysql-connector-python
apt-cache search MySQLdb
sudo apt-get install python-mysqldb
```

Una vez instalados los paquetes necesarios se ha desarrollado el código correspondiente a este cliente, el cual se puede encontrar en el Anexo II situado a final del documento. En dicho código se integra la recopilación de datos por parte del sensor DHT11 y su publicación a través del protocolo MQTT.

3.1.2. Broker MQTT

El broker MQTT se trata de un elemento fundamental para llevar a cabo el envío de los datos entre los clientes. Es necesario que este permanezca en constante funcionamiento y sea accesible desde una red externa.

El objetivo es que los clientes MQTT no se vean limitados a estar en la misma red en la que se encuentra el broker, por lo que se ha optado por hacer uso de un servidor MQTT público debido a que una de las premisas de este trabajo final de grado es implementar una solución de bajo coste y con herramientas de desarrollo de código abierto.

En la propia página web de MQTT se puede encontrar una lista con los broker de acceso público disponibles para usar este protocolo. Estos, generalmente son utilizados para la realización de pruebas y prototipos, y entre ellos se distinguen tanto brokers gratuitos como de pago.

En la Tabla 1 se muestran dos broker de tipo Mosquitto que se han seleccionado entre los disponibles, así como una comparativa de sus capacidades:

	Dirección	**Puertos**	**Otras características**
Mosquitto Broker	test.mosquitto.org	1883: no encriptado 8883: encriptado 8884: encriptado, requiere certificado del cliente 8080: MQTT sobre WebSockets, no encriptado 8081: MQTT sobre WebSockets, encriptado	▪ Servicio gratuito destinado a la realización de pruebas. ▪ No ofrece garantía de disponibilidad en todo momento. ▪ No indica un límite en el número de clientes/conexiones.

Cloud MQTT	En este servicio el usuario puede crear varias instancias según el plan elegido. Cada vez que se crea una nueva, la dirección real del broker varía en función de la región y centro de datos elegido. A su vez el puerto a utilizar que se le indica al usuario varía cada vez que se crea una instancia nueva.	▪ Servicio de pago, lo que aumenta considerablemente su disponibilidad. ▪ Ofrece un plan gratuito: - Velocidad de transmisión de información: 10kbit/s. - Máximo de 5 conexiones/clientes. - Se proporciona un usuario y contraseña para el cliente MQTT.

Tabla 1. Comparación de brokers públicos tipo Mosquitto [18]

Otras opciones de broker MQTT de tipo Mosquitto disponibles son: *mqtt.eclipse.org* (con características muy similares a *test.mosquitto.org)* y *mqtt.swifitch.cz* (destinado al proyecto Swifitch pero también disponible para pruebas).

De todas ellas se ha decidido hacer uso de *test.mosquitto.org*, debido a que no ofrece limitaciones y permite usar el protocolo MQTT encriptado.

Para el uso del protocolo encriptado a través del puerto 8883 ha sido necesario descargar el certificado correspondiente desde la página de este broker[2], donde se indica que los puertos cifrados admiten las versiones TLS v1.3, v1.2 y v1.1 con certificado x509 y se requiere soporte del cliente para la conexión. Los comandos necesarios para ello quedan reflejados en los Anexos II y III, donde se detallan los códigos usados para ambos clientes MQTT.

3.1.3. Sistema del nodo servidor

Este nodo cuenta con dos funciones:

- Servidor, en el que se alojan tanto la base de datos como los ficheros que componen la aplicación web. Accesible para los usuarios finales.
- Cliente MQTT subscrito a los temas en los que se envían los datos de interés. A su vez, el *script* que se ha creado para este cliente guarda esta información en la base de datos correspondiente.

3.1.3.1. Servidor

3.1.3.1.1. Servidor Web

Como se ha visto en el apartado 3.1.1., el nodo cliente está formado por un sensor conectado a una Raspberry Pi 3 B+, a la cual se le ha configurado Raspbian (distribución de Linux basada en Debian). Debido a esto y para evitar posibles incompatibilidades, se ha decidido usar Linux como sistema operativo para el servidor web, optando entre todas sus distribuciones por Debian.

A causa de que el nodo servidor de este sistema se ha decidido alojar en un terminal con sistema operativo Windows 10, se necesitará hacer uso de una máquina virtual para evitar realizar una

[2] http://test.mosquitto.org/

partición del disco. Para ello se ha escogido el software VirtualBox, el cual se puede descargar desde su misma página web[3].

Tras la instalación del software, a la máquina virtual creada se le han asignado las siguientes propiedades:

RAM	1024 MB
Tipo de archivo de disco duro	VDI (VirtualBox Disk Image)
Almacenamiento en unidad de disco duro física	Tamaño Fijo: 20GB

Tabla 2. Propiedades de la máquina virtual creada

Una vez creada, se ha configurado el sistema operativo deseado, descargando el archivo necesario desde la página web de Debian[4] y cargándolo en *Configuración* → *Almacenamiento* → *Controlador: IDE* → *Vacío*. Por otro lado, en el apartado de *Red* se ha seleccionado la opción *Adaptador puente*.

Al configurar estos parámetros se ha iniciado la máquina virtual. Gracias al instalador gráfico de Debian, solo ha sido necesario seguir las pautas indicadas para disponer del sistema operativo funcionando correctamente en la máquina.

Tras estos pasos, se llega al momento de la elección del servidor que se va a utilizar. Actualmente existe una gran variedad de servidores en el mercado, por lo que elegir una opción no es una tarea sencilla. Debido a que la finalidad de este servidor es proporcionar almacenamiento, procesamiento, y consultas de datos de las aplicaciones de IoT, se deberá tener en cuenta requerimientos importantes de un sistema IoT, tales como: admitir un número alto de conexiones simultáneas y flexibilidad.

La Figura 18 muestra una gráfica con información de los servidores más utilizados, situados en función de la cantidad de sitios web que hacen uso de ellos y a su vez el tráfico que procesan. Cabe mencionar que en ella también se ha añadido Lighttpd, ya que, aunque no fuese uno de los más empleados, se vio que podía ofrecer características interesantes para este caso de uso. Entre los servidores que se muestran en dicha gráfica, posteriormente no se considerarán los que ofrecen su versión completa de pago (Cloudflare Server, Microsoft-IIS, LiteSpeed), ya que se desea implementar una solución de bajo coste.

[3] https://www.virtualbox.org/
[4] https://www.debian.org/index.es.html

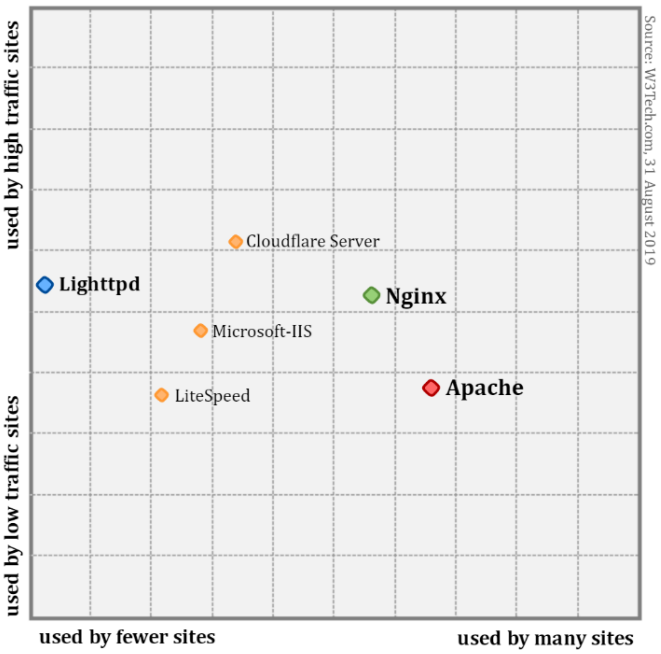


Figura 18. Servidores en términos de popularidad y tráfico [19]

Por tanto, los servidores seleccionados que se van a comparar son Apache, Nginx y Lighttpd. En la Figura 19 se muestra el porcentaje de uso de cada uno de dos formas diferentes:

a) La figura de la izquierda representa el porcentaje de uso de estos servidores considerando diferente número de sitios web en cada caso (Global, Top 1 Millón, Top Cien Mil, Top Diez Mil y Top Mil).

b) La figura de la derecha se muestra la evolución del porcentaje de sitios web hacen uso de estos servidores a lo largo del último año.

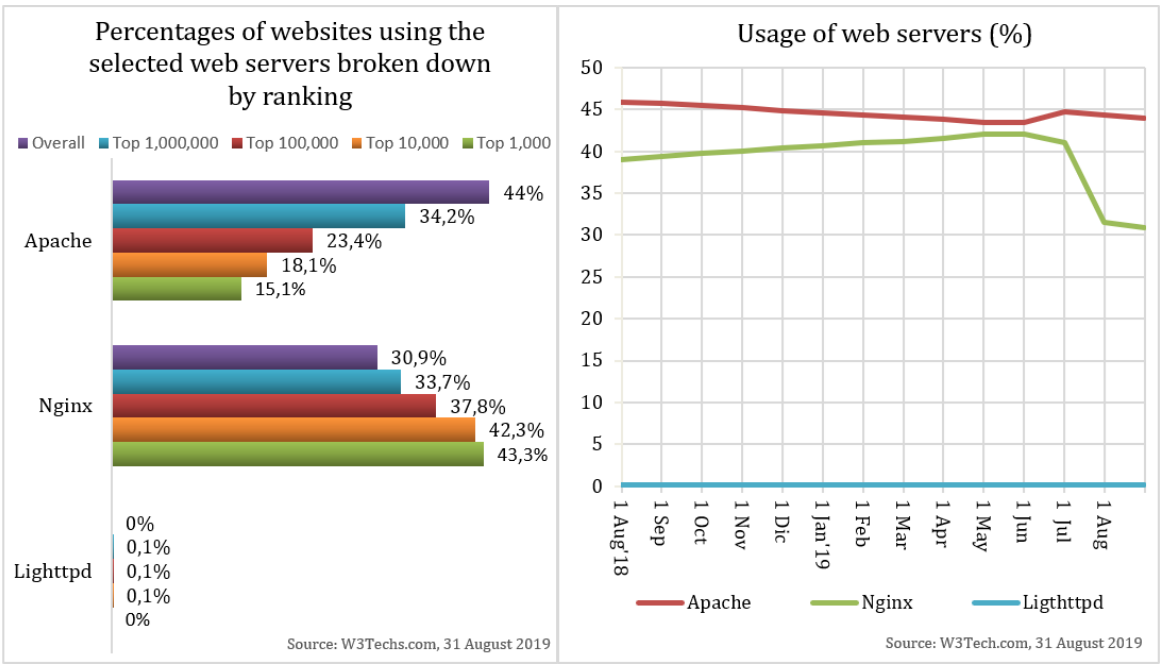


Figura 19. Porcentaje de uso de los servidores [20]

Tal y como se muestra en la Figura 19, observamos que Apache es el servidor web más utilizado globalmente, sin embargo, como refleja la Figura 18 no es la mejor opción para una solución de

IoT, puesto que de los tres es el que se posiciona en un lugar más bajo respecto al tráfico que procesa. En este aspecto Nginx y Lighttpd muestran resultados muy parecidos, por lo que ambos podrían ser opciones válidas para este caso.

Como se ve, la Figura 18 solo aporta que Lighttpd se usa en un 0.1% de los casos. Aunque se pueda pensar que Nginx puede ser la mejor opción, este no es un dato decisivo, por lo que para decantarse por uno de los dos se han de comparar sus características con algo más de profundidad. En la Tabla 3 se puede encontrar un resumen de esta información.

	Características generales
Apache[5]	- Open source. Gratuito. - Arquitectura bloqueante. Basado en hilos de ejecución, por lo que presenta un bajo rendimiento al recibir miles de peticiones simultáneas. - Mayor consumo de memoria.
Nginx[6]	- Open Source. Gratuito. - Servidor rápido y ligero. - Arquitectura basada en eventos (asíncrona), lo que permite gestionar miles de conexiones simultáneas consumiendo menos recursos que otros servidores. Alto rendimiento. - Aborda el problema C10K (optimización de conexiones para administrar gran número de conexiones concurrentes). - Proxy inverso.
Lighttpd[7]	- Open Source. Gratuito. - Rápido y ligero. - Arquitectura orientada a eventos (no bloqueante), optimizado para entornos de alto rendimiento, consumiendo pocos recursos. Resuelve problema C10K. - Comunidad menor a la de Nginx.

Tabla 3. Comparación de los servidores considerados [21]

En cuestiones de seguridad y contenido dinámico, las tres opciones presentan características muy similares. Igualmente, los tres son soportados por gran variedad de sistemas operativos, incluyéndose Linux entre ellos.

Por otra parte, como se venía intuyendo gracias a la Figura 18 y como se refleja en la Tabla 3, aunque Apache sea uno de los servidores más utilizados, para este caso no es la mejor opción, ya que presenta el problema de no poder procesar gran cantidad de conexiones simultáneas debido a la arquitectura que presenta.

Si comparamos Nginx y Lighttpd en este sentido, presentan características muy similares, diferenciándose en pequeños detalles. Por ello se puede concluir que cualquiera de los dos es una buena opción para el sistema que se pretende implementar. Para decantarse por uno, se ha

[5] https://httpd.apache.org/
[6] https://www.nginx.com/resources/wiki/
[7] https://www.lighttpd.net

valorado que la comunidad y documentación que presenta Nginx es mucho mayor, ya que su porcentaje de uso es mucho más elevado.

Tras haber seleccionado Nginx como el servidor web a utilizar en este sistema, se ha procedido a su instalación. Para ello se han hecho uso de los comandos mostrados a continuación, desde la Terminal de Debian. Al no haber fijado una contraseña de *superusuario* a la hora de configurar el sistema operativo, este no se ha creado, por lo que para contar con los privilegios necesarios se debe añadir el comando sudo en cada instrucción.

```
sudo apt-get update (Actualización del sistema)
sudo apt-get upgrade
sudo apt-get install nginx (Instalación de Nginx)
sudo systemctl enable nginx (Habilitar Nginx)
(Las siguientes instrucciones no son estrictamente necesarias para la
instalación, pero son de utilidad)
sudo systemctl restart nginx (Restart Nginx)
sudo systemctl stop nginx (Stop Nginx)
sudo systemctl start ngnix (Start Nginx)
sudo systemctl status nginx (Comprobar )
```

En este punto, se puede comprobar que la instalación se ha realizado con éxito simplemente escribiendo en cualquier navegador la dirección IP privada de la máquina en la que se ha instalado Nginx.

Como se desea que este servidor sea accesible desde cualquier terminal que se encuentre en una red externa a la de dicho servidor, se ha procedido a abrir el puerto 80[8] para la IP privada correspondiente a la máquina donde se encuentra el servidor (Figura 21). Por esta razón antes de todo, se ha tenido que fijar esta dirección (Figura 20), ya que de lo contrario iría cambiando debido al protocolo DHCP que asigna dinámicamente las direcciones en la red.

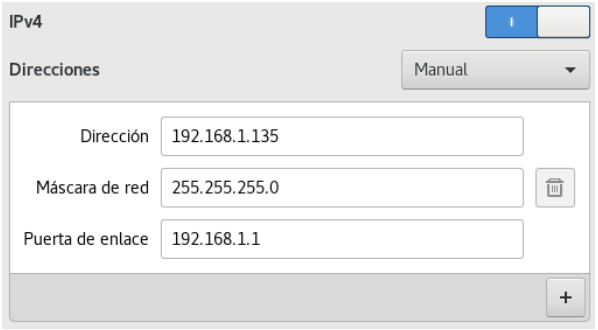


Figura 20. Configuración de una dirección IP fija en el servidor

Enable	Name	WAN Host Start IP Address	WAN Start Port	LAN Host Start Port	WAN Connection	Modify	Delete
	Protocol	WAN Host End IP Address	WAN End Port	LAN Host End Port	LAN Host Address		
✓	pm_1		80	80	WANConnectio		
	TCP AND I		80	80	192.168.1.135		

Figura 21. Apertura del puerto 80 en el router

Una vez realizado este paso, el servidor ya sería accesible desde cualquier terminal, para ello solo habría que escribir en un navegador la IP pública que identifica a la red desde el exterior.

[8] Este es el puerto en el servidor web escucha las peticiones HTTP de los clientes.

Para facilitar el acceso a los usuarios a la aplicación web y debido a que no se posee un dominio propio se ha hecho uso de Duck DNS[9], un servicio gratuito que apunta un nombre de dominio a la IP que se elija, es decir la IP pública de la red en la que se halla el servidor.

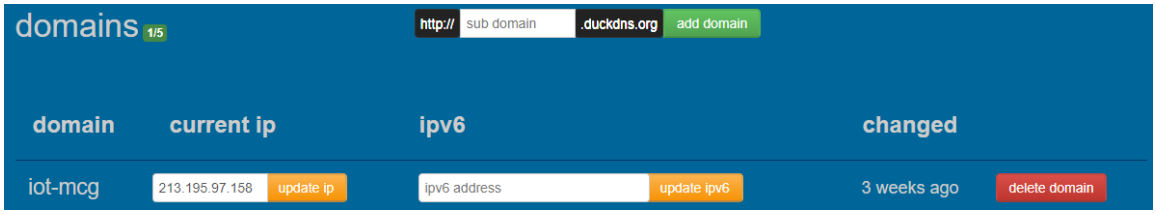


Figura 22. Configuración del nombre de dominio con Duck DNS

3.1.3.1.2. Base de Datos MySQL

Como se ha mencionado anteriormente en este documento, existe la necesidad de almacenar los datos medidos por el sensor, por ello se ha escogido MySQL como sistema de gestión de base de datos relacional a utilizar. MySQL es una de las bases de datos de código abierto más populares en entornos de desarrollo web, por lo que la compatibilidad con otras herramientas que se han de utilizar en este sistema es mayor.

Para ello, previamente se ha descargado el paquete .deb necesario desde el apartado *APT Repository* de su página oficial[10]. Para la instalación de dicho paquete, ha de ejecutarse en una Terminal el siguiente comando:

```
sudo dpkg -i mysql-apt-config_0.8.11-1_all.deb
```

Por otra parte, se ha decidido usar phpMyAdmin para la gestión de las bases de datos MySQL. Esta herramienta escrita en PHP facilita la administración ofreciendo una interfaz gráfica a través de un navegador web. Los comandos que se deben introducir en la Terminal para la utilización de phpMyAdmin serían:

```
sudo apt upgrade
sudo apt install phpMyAdmin
sudo dpkg-reconfigure phpMyAdmin
rm -rf /usr/share/nginx/www (Borrar enlace por defecto)
sudo ln -s /usr/share/phpMyAdmin/ /var/www/html/phpMyAdmin (Crea un enlace
nuevo)
```

Al abrirse el configurador de phpMyAdmin, se han de seguir los pasos indicados. Una anotación importante es que a la hora de seleccionar el servidor web que se va a utilizar solo se muestran los visibles en la Figura 23, por lo que para poder hacer uso de Nginx simplemente se tendrá que dejar esta opción en blanco.

[9] https://www.duckdns.org/
[10] https://dev.mysql.com/downloads/

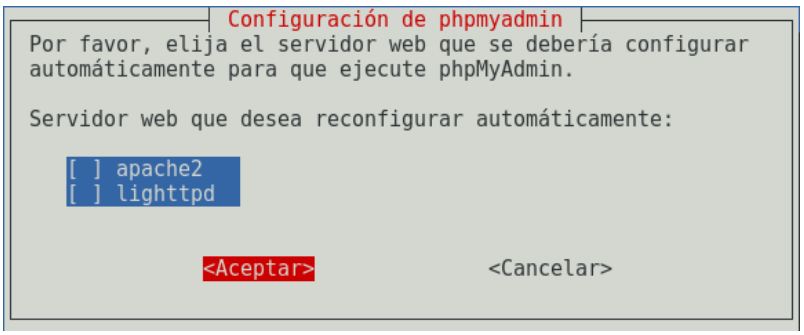


Figura 23. Elección de servidor en la configuración de phpMyAdmin

Tras esto, ya se puede crear la base de datos que se aloja en el servidor introduciendo en cualquier navegador: *ip_privada_de_la_máquina/phpmyadmin* (Ej.: 192.168.1.135/phpmyadmin).

La estructura que se ha fijado para la base de datos de este proyecto, así como las relaciones entre los campos de las distintas tablas que la forman se refleja en la Figura 24.

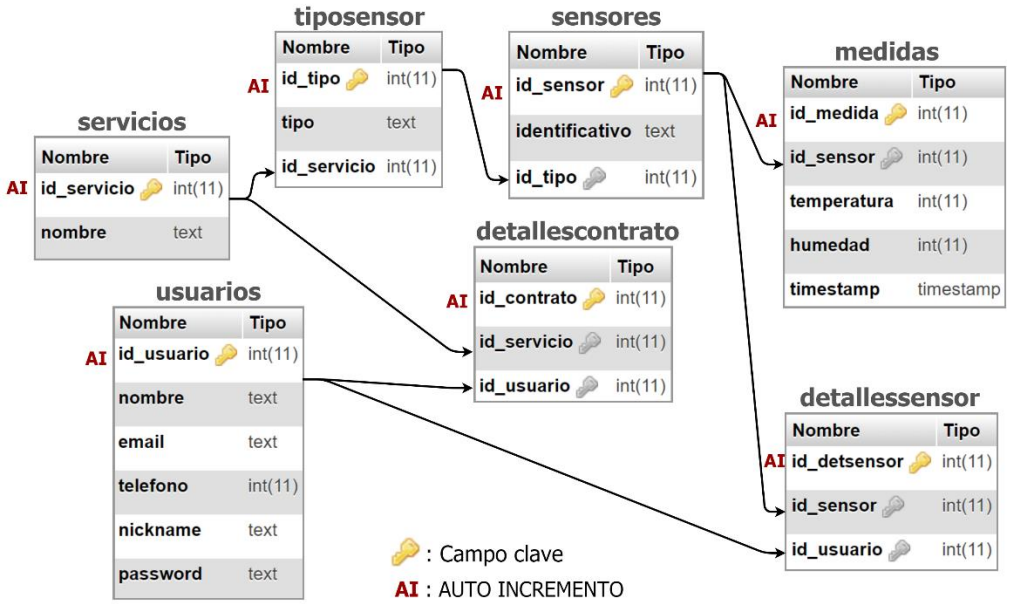


Figura 24. Estructura de la base de datos

Tal y como se muestra en la figura anterior, se han definido 7 tablas para la estructura de almacenamiento y control de los datos que se capturan de los usuarios IoT, para su posterior procesamiento, visualización y análisis:

La **tabla *usuarios*** almacena la información con la que se registran en la aplicación web. Se han definido identificadores como: *id_usuario*; *nombre*; *email*; *teléfono*; *nickname*; *password*. Por otro lado, se tienen las tablas de ***servicios*** y ***sensores***, en las que se guardan datos relativos a los diferentes servicios que se ofrecen en la plataforma y los sensores que se encuentran en activo, respectivamente.

Cada uno de los sensores se identifica con un tipo de la **tabla *tiposensor***, correspondiendo a su vez cada tipo a uno de los servicios disponibles. La **tabla *detallescontrato***, relaciona cada uno de los servicios con los usuarios o clientes que lo hayan solicitado o contratado, y ***detallessensor*** enlaza los sensores con los usuarios que tienen acceso a las medidas que realizan estos.

Por último, se encuentra la **tabla de *medidas***, en la que se guarda la información recibida desde el sensor mediante el protocolo MQTT. Debido que solo se dispone de un sensor de temperatura y humedad, los datos almacenados en esta tablan irían orientados en especial al servicio de *Smart*

Agriculture. Es importante resaltar que la estructura de la base de datos se ha diseñado de forma que sea escalable conforme se vayan configurando más servicios IoT, no obstante, se deberán hacer pequeños ajustes en la tabla de *medidas*.

En cuanto a las conexiones existentes entre las tablas que conforman la base de datos, en la Figura 24. quedan reflejadas mediante flechas. Como se puede apreciar, todas las relaciones existentes entre tablas se han efectuado con campos clave pertenecientes a otras tablas. Esta es una variable de tipo entero única para cada registro de la tabla a la que pertenece que se incrementa con cada nueva fila introducida por lo que no se repite. Para hacer más sencilla la comprensión de estas relaciones se ha utilizado el mismo nombre tanto en el campo clave de la tabla original como en el campo que se quiere relacionar en la otra tabla. Esto no es estrictamente necesario, podría establecerse la misma relación utilizándose nombres diferentes.

3.1.3.1.3. Aplicación Web

Una vez se tiene MySQL y phpMyAdmin correctamente funcionando y por consiguiente se ha creado la base de datos necesaria para el almacenamiento de la información, quedaría por desarrollar la aplicación web a la que accederá el usuario final IoT que haga uso de esta plataforma.

Los ficheros correspondientes a dicha página web estarán escritos en lenguaje HTML y PHP, haciendo uso también de hojas de estilo (CSS).

Por un lado, el uso de HTML (*HyperText Markup Language*, en inglés) está más que justificado, ya que se trata de un lenguaje de marcas de hipertexto, como sus siglas indican, destinado al desarrollo de páginas de Internet. Este lenguaje, junto a CSS, para simplificar el código, nos permitirá definir el diseño gráfico de la web de forma estructurada y ordenada (el tipo de letra para el texto, los colores o imágenes de los enlaces o el fondo de la página entre otros efectos).

Sin embargo, no se quiere una página web estática, ya que entre los objetivos de la aplicación se encuentra el registro e inicio de sesión de usuario, la representación de gráficas y estadísticas con los datos recogidos por los sensores en tiempo real, es decir, se necesita que la web sea interactiva. Por esta razón se debe hacer uso de un lenguaje de programación con el que conectar a nuestra base de datos y obtener información referente a los usuarios, servicios y sensores.

Entre todos los posibles lenguajes de programación que existen y permitirían cumplir estos requisitos se ha escogido PHP, un lenguaje libre y abierto adecuado para el desarrollo web. Algunas de las características por las que se ha escogido este lenguaje son:

- Lenguaje totalmente libre y abierto. Multiplataforma.
- Presenta una sintaxis sencilla y puede ser fácilmente incrustado en HTML.
- Orientado al desarrollo de páginas web dinámicas, presentado capacidad de conexión con la mayoría de los sistemas de gestión de bases de datos. Destacando especialmente por su conectividad con MySQL.
- Código fuente invisible al cliente, ya que este se ejecuta desde el lado del servidor.

Normalmente se presenta como desventaja de PHP que el código no es ocultado en el servidor en el que se ejecuta, lo que podría ser un inconveniente en cuanto a seguridad si se aloja en un servidor no propietario. Sin embargo, en este caso, sí se dispone de un servidor propio para ejecutar la aplicación por lo que esto no supone realmente un inconveniente.

Para poder ejecutar en el servidor Nginx el código PHP escrito para que la página web sea dinámica, se necesitan dos pasos previos. El primero de ellos es instalar el paquete *php-fpm* (*FastCGI Process Manager*). Esto se debe a que Nginx no incluye de forma nativa el procesamiento de PHP. También se ha de instalar el paquete *php-mysql* que ayuda a comunicar

PHP con la base de datos MySQL. La instalación de ambos se consigue escribiendo el siguiente comando:

```
sudo apt install php-fpm php-mysql
```

En segundo lugar, hace falta modificar el fichero de configuración del servidor, el cual se encuentra en: *Equipo* → *etc* → *nginx* → *sites-available* → *default*. En este fichero se debe añadir "index.php" para que los archivos PHP sean interpretados, como se observa en la Figura 25.

```
root /var/www/html;

# Add index.php to the list if you are using PHP
index index.php index.html index.htm index.nginx-debian.html;

server_name _;

location / {
        # First attempt to serve request as file, then
        # as directory, then fall back to displaying a 404.
        try_files $uri $uri/ =404;
}

# pass PHP scripts to FastCGI server
#
location ~ \.php$ {
        include snippets/fastcgi-php.conf;
#
#       # With php-fpm (or other unix sockets):
        fastcgi_pass unix:/var/run/php/php7.0-fpm.sock;
#       # With php-cgi (or other tcp sockets):
#       fastcgi_pass 127.0.0.1:9000;
}
```

Figura 25. Configuración del servidor Nginx

Concluidos estos pasos, ya se puede ejecutar correctamente el código de la página web escrito en PHP, así como realizar la conexión con la base de datos MySQL y hacer consultas, obteniendo datos almacenados y/o creando nuevos registros en las tablas.

Para el desarrollo de la aplicación web, como se ha dicho, se ha utilizado HTML junto a CSS para definir la estructura y diseño de las diferentes pantallas que la conforman, integrando fácilmente PHP para conseguir así una página web dinámica. Además, para conseguir ciertos aspectos de la página web, como puede ser actualizar determinadas partes sin que el usuario tenga que recargar la página o contar con diferentes representaciones gráficas de los datos que ofrecen los servicios de IoT a los que se subscribe el usuario, se ha hecho uso de algunas librerías de código abierto de JavaScript. A continuación, se describe brevemente la utilización de cada una de ellas:

Librerías	Aplicación de uso
slider.js *jquery.min.js*[11]	*slider.js* no se trata de una librería externa de JavaScript, sino que es un fichero propio que se ha escrito con el fin de lograr la galería de imágenes que se muestran al inicio. Para ello es necesaria también la librería *jquery.min.js*.
prototype_1.6.0.2.js[12]	Se hace uso de esta librería para conseguir un formulario de registro que compruebe en el acto que no existe un nombre de usuario igual al que el nuevo cliente desea utilizar, avisándole así si el nombre escogido se encuentra disponible o no.
jquery.min.js	En este caso se ha hecho uso de *jquery.min,.js* para lograr actualizar distintas partes de la aplicación web como son: los valores instantáneos de temperatura y humedad recogidos o la lista de servicios del usuario una vez que solicita el alta o baja de alguno de ellos.
chart.js[13]	Esta librería permite la representación de las gráficas en de temperatura y humedad haciendo uso de la etiqueta <*canvas*> de HTML.
jquery.easypiechart.js[14]	La utilización de esta librería permite realizar un gráfico circular con animación en el que se representa el último valor que ha tomado el sensor del porcentaje de humedad.

Tabla 4. Librerías JavaScript utilizadas

Una vez aclarados los lenguajes y librerías necesarias para el desarrollo de la aplicación web, se explicará la estructura de la aplicación web que se pone a disposición del usuario para la visualización de los datos recogidos por los sensores para los servicios IoT que se ofrecen en diferentes ámbitos. Además de su estructura se comentarán los detalles más importantes en la programación de esta aplicación, adjuntando el código correspondiente.

En la Figura 26 se muestra un diagrama de fujo de las actividades o funcionalidades con las que cuenta esta aplicación web. El desarrollo de esta aplicación web se ha orientado a que la solución implementada tenga carácter comercial, por ello se ofrece la posibilidad al usuario final de registrarse, modificar sus datos personales, solicitar el alta o baja de servicios y visualizar la información relativa a ellos.

[11] https://jquery.com/download/
[12] http://prototypejs.org/download/
[13] https://github.com/chartjs/Chart.js
[14] https://cdnjs.cloudflare.com/ajax/libs/easy-pie-chart/2.1.4/jquery.easypiechart.js

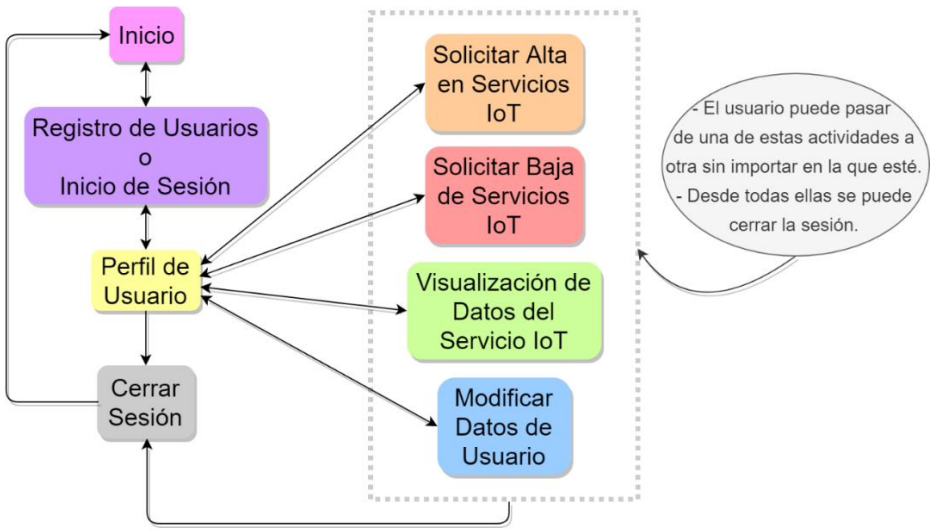


Figura 26. Diagrama de flujo de actividades de la aplicación web

Como se puede intuir, el diagrama de la Figura 26 no hace referencia a los ficheros que se han necesitado escribir, sino a las diferentes funciones que se ofrecen al usuario dentro de la aplicación, pudiendo estar cada una de ellas compuesta por uno o más ficheros.

A continuación, se describen cada una de estas funciones y el código necesario de los elementos más característicos que las componen. Como se ha mencionado con anterioridad, la mayor parte del diseño gráfico de la web se ha llevado a cabo gracias a HTML y CSS, por lo que en los siguientes puntos la descripción se centrará sobre todo en la parte dinámica de la aplicación desarrollada con PHP en la que se han llevado a cabo las consultas a la base de datos necesarias.

- **Inicio**

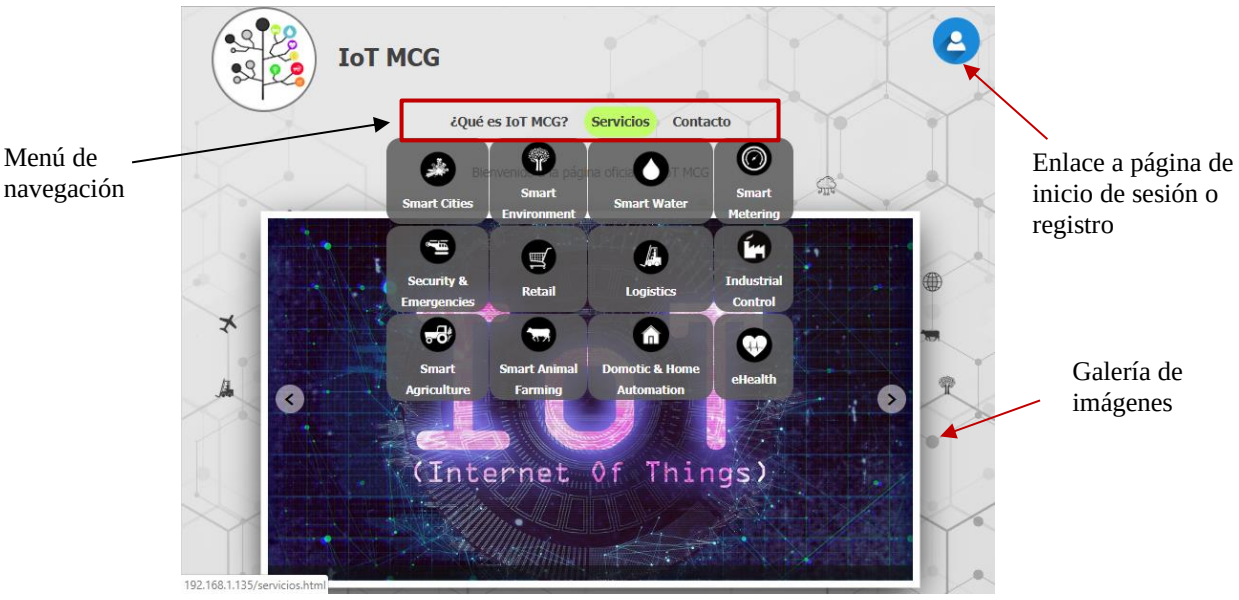


Figura 27. Pantalla de inicio

La figura anterior muestra la página de inicio que se presenta al usuario (*index.html*), donde se han señalado los elementos más relevantes que la componen.

Se ha creado un menú de navegación en el que se iluminan las opciones al pasar el cursor por encima de ellas y en el caso de la opción *Servicios* también aparece una tabla con las diferentes alternativas ofrecidas. Estas opciones del menú consisten en enlaces a diferentes páginas donde

se puede encontrar una descripción de la aplicación, de los servicios IoT disponibles o los datos de contacto. Este menú se ha creado en el código HTML gracias al uso de las etiquetas *<nav></nav>*, entre las que se han introducido los elementos deseados. Por otra parte, el estilo y formato de este menú se ha conseguido con CSS.

Para que la página inicial se vea más atractiva hacia el usuario se ha optado por programar una galería de imágenes que tiene como objetivo el presentar la aplicación web y mostrar algunos de los aspectos que se ofrecen al cliente. Para conseguir esto, se ha hecho uso de la librería *jquery.min.js* de JavaScript y se ha escrito un fichero en este mismo lenguaje al que se ha nombrado *slider.js*. Ambos códigos deben ser importados dentro del archivo *index.html* de la siguiente manera:

```
<script src="jquery.min.js" type="text/javascript"></script> (Librería
jquery.min.js)
<script src="slider.js"></script> (Fichero slider.js)
```

La galería se ha definido en *index.html* con las etiquetas * *. Dentro de ellas se encuentran tantas secciones *(<section>...</section>*) como imágenes componen dicha galería, definiendo en el interior de estas la imagen y el texto que se quiere mostrar. Un fragmento de dicho código sería el siguiente:

```
<div id="slider">
        <section>
                <img src="images/capt2.png" alt="">
                <div class="numbertext"></div>
                <div class="text"><p></p></div>
        </section>
        ...
</div>
```

En este caso, en las divisiones *numbertext* y *text* no se ha escrito texto o número de diapositiva, ya que el ejemplo hace referencia a la primera imagen de la galería que podría interpretarse como una portada.

Además, se definen los botones que se utilizarán para deslizar las diapositivas manualmente de la siguiente forma:

```
<div id="btn-prev">&#60;</div>
<div id="btn-next">&#62;</div>
```

El código del archivo *slider.js*, por tanto, se centra en conseguir que las imágenes definidas sean capaces de ir rotando, bien de manera automática cada cierta cantidad de segundos o de manera manual. Este código se adjunta en el apartado A del Anexo IV.

Para finalizar con este apartado, hay que mencionar que se ha colocado un enlace, al que se le ha asignado un icono para hacerlo más visual, que permite al usuario acceder a la página de registro o inicio de sesión.

- **Registro de Usuarios o Inicio de Sesión**

Esta pantalla de la aplicación web presenta al usuario dos opciones, el inicio de sesión si ya dispone de un usuario o el registro en caso contrario. Ambas opciones se tratan de formularios independientes, lo que queda reflejado en la siguiente figura:

Figura 28. Pantalla de registro o inicio de sesión

Para el **registro** de usuario se han definido seis campos, cinco de los cuales serán almacenados en la tabla *usuarios* de la base de datos. El sexto campo se trata de replicar la contraseña elegida, con el fin de verificar la contraseña introducida por el usuario. El valor introducido en estos campos es enviado al archivo *registro.php*, que es el que se encarga de crear el registro del nuevo usuario.

Lo primero que se realiza en dicho fichero es iniciar una sesión PHP y conectar con la base de datos para posteriormente realizar las consultas correspondientes. En este caso también es necesario obtener los datos que el usuario ha introducido en el formulario. El código correspondiente a lo explicado sería el siguiente:

```php
<?php
        session_start(); //Inicio de sesión PHP
        $s="127.0.0.1"; //Dirección del servidor (se ejecuta en local)
        $bd="iotservices"; //Nombre de la base de datos
        $u="root"; // Usuario de la base de datos
        $p="Iot.Mysql"; // Contraseña
        $conexion = new mysqli($s,$u,$p,$bd); //Conexión con la base de datos
        // Obtención de los datos del formulario por el método POST
        $name           = $_POST['name'];
        $email          = $_POST['email'];
        $phone          = $_POST['phone'];
        $nick           = $_POST['nick'];
        $pass           = $_POST['pass'];
        $rpass          = $_POST['rpass'];
        ...
?>
```

Dentro de este código PHP es donde se realizarán la consulta que creará el nuevo registro, tras comprobar con condicionales que las contraseñas introducidas son iguales y que no existe un registro con el nombre de usuario elegido:

```
//Consulta para el registro de un nuevo usuario
$sql2 = "INSERT INTO usuarios (id_usuario, nombre, email, telefono, nickname,
password) VALUES (null,'$name','$email', '$phone','$nick','$md5pass')";
```

El *id_usuario* se autoincrementa, por lo que al crear un registro se indica el valor *null*.

Hay que decir, que antes de insertar los datos, se aplica el algoritmo MD5 a la contraseña para no almacenarla como texto plano.

Por otro lado, para que esta y las demás consultas que se realizan se ejecuten, se necesita del siguiente código:

```
$mysqli_query($conexion, $sql2); //Ejecución de la consulta SQL
```

En este caso no es necesario, ya que la consulta inserta un nuevo registro, pero si se tratase de un *SELECT*, en el que se obtienen datos, habría que añadir al código PHP también:

```
$result = mysqli_fetch_array($datos); //Array resultado de la consulta
```

Y en el caso en el que se quisiese saber el número de registros devueltos por la consulta:

```
$cuenta = mysqli_num_rows($datos); //Recuento de registros devueltos tras la
consulta
```

Como se ha mencionado al explicar las librerías de JavaScript utilizadas, en el mismo formulario se ha hecho uso de un script utiliza una de estas librerías, el cual llama al fichero *check_nick.php* que consulta si existe un nombre de usuario igual al escogido e informa al usuario en el momento. Este código se adjunta en el apartado B del Anexo IV.

En cuanto al formulario de **inicio de sesión**, se cuenta con solo dos campos que son los suficientes para identificar al usuario, estos valores introducidos por el usuario son enviados al fichero *inicio.php*, donde se verifica que existe un usuario en la base de datos que coincide con estos parámetros y se inicia su sesión. De forma similar al caso del registro, lo primero es iniciar una sesión PHP y conectar con la base de datos, así como obtener los datos del formulario mediante el método *$_POST*. Tras esto, se realizan las consultas oportunas, que en este caso serían:

```
//Obtención de los datos de usuario para almacenamiento en variables de sesión
$sql1 = "SELECT id_usuario, nombre, email, telefono, nickname, password  FROM
usuarios WHERE nickname = '$nick' and password ='$md5pass' ";

//Consulta de servicios activos del usuario (almacenamiento en variables de
sesión)
$sql2 = "SELECT servicios.nombre, servicios.id_servicio FROM servicios INNER
JOIN detallescontrato ON servicios.id_servicio = detallescontrato.id_servicio
INNER JOIN usuarios ON detallescontrato.id_usuario = usuarios.id_usuario WHERE
usuarios.nickname= '$nick' AND usuarios.password='$md5pass'";
```

Tras iniciar sesión, se utilizan variables de sesión PHP para pasar los parámetros relativos al usuario y tenerlos disponibles en el resto de las páginas siguientes, sin tener que repetir las consultas a la base de datos. Un ejemplo de variable de sesión es el que se muestra a continuación:

```
<?php
        session_start(); //Inicio de sesión PHP
```

```
        ...
        $_SESSION['nickname'] = $result['nickname']; // Variable de sesión  en
la que se guarda un valor resultado de una consulta
        ...
?>
```

Para que esta variable pueda realmente disponible en las páginas en las que se necesite, dentro del código PHP habrá que obtenerla de la siguiente forma:

```
<?php
        session_start(); //Mantiene sesión iniciada
        $usuario =$_SESSION['nickname'];
        ...
?>
```

Además, al igual que en la anterior pantalla, se ha agregado un enlace con un icono (esquina superior derecha de la figura anterior) para poder retornar a la página de inicio fácilmente.

- **Perfil de Usuario**

Una vez que el cliente se ha autenticado con su nombre de usuario y contraseña, accede a su perfil, donde se muestran los datos personales con los que se registró y los servicios activos que dispone, también se le brinda la posibilidad de realizar ajustes en su cuenta accediendo a la página correspondiente a través del enlace que se indica en la Figura 29.

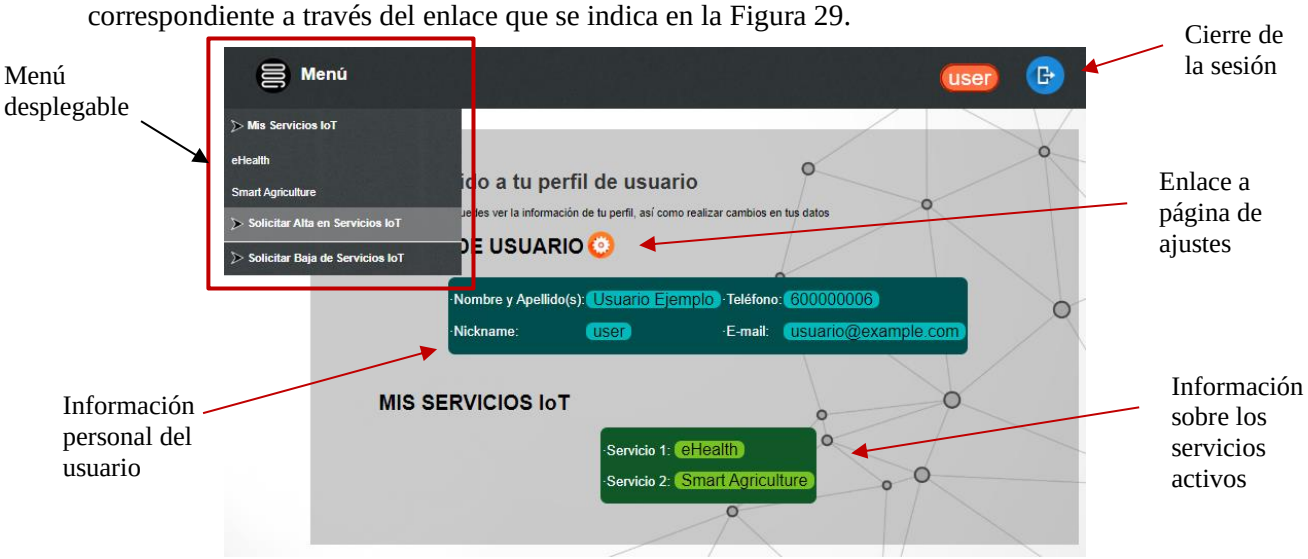


Figura 29. Perfil de usuario

A partir de este punto, se tienen dos elementos comunes a este y los siguientes bloques:

- **Cierre de sesión**: situado en la esquina superior derecha de la página. Al hacer clic sobre este icono, se retorna a la página de inicio cerrando la sesión de usuario que se encontraba abierta, por lo que las variables de sesión PHP ya no estarán disponibles. Para conseguir esto, este enlace ejecuta el fichero *salir.php*, que se compone del siguiente código:

```
<?php
        session_start();
        session_destroy();
        header('Location: index.html');
?>
```

- **Menú desplegable**, el cual se ha realizado exclusivamente con código HTML y dándole el estilo deseado con CSS. Se trata de una lista con enlaces a los servicios IoT activos del usuario (acceso a su área de visualización de datos) y a la solicitud de alta o cancelación de los servicios.

Por otra parte, la información relativa al usuario que se muestra en esta página (tanto los datos personales, como los servicios activos del cliente) se obtiene de las variables de sesión que se habían declarado en el archivo *inicio.php*.

- **Modificar Datos de Usuario**

En esta pantalla se brinda la posibilidad al usuario de modificar los datos personales con los que se registró inicialmente, realizar un cambio de contraseña o incluso eliminar su cuenta en el caso de que ya no desee hacer uso de la plataforma. Como se ve en la Figura 30 se pueden diferenciar tres secciones correspondientes a las acciones que se han citado.

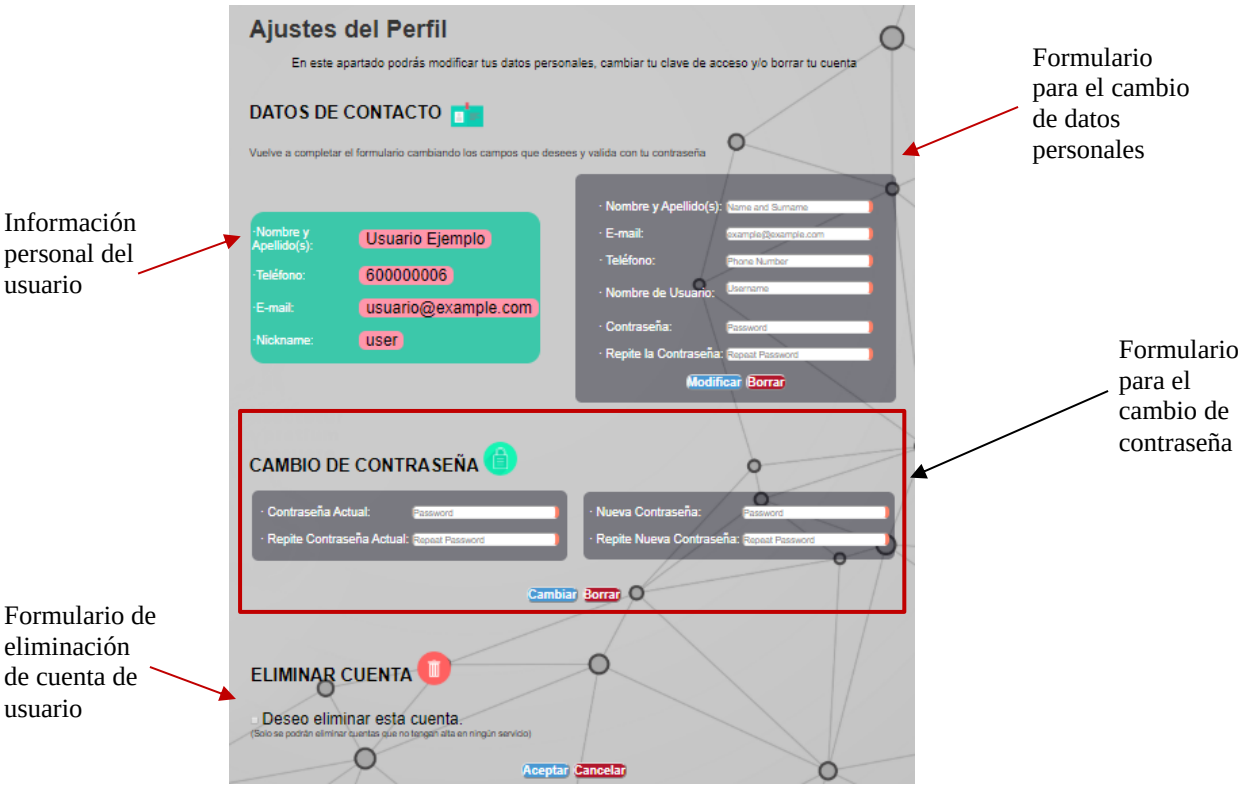


Figura 30. Ajustes de perfil de usuario

En la sección de *DATOS DE CONTACTO*, se muestra a la izquierda la información personal actual del usuario, mientras que a la derecha se dispone de un formulario similar al que se tenía en la página de registro, realizando prácticamente la misma consulta, pero en este caso el introducir la contraseña tiene la función de validarla con la que se tenía guardada en la tabla de *usuarios.*

```
$sql2 = "UPDATE usuarios SET nombre = '$name', email= '$email',
telefono='$phone', nickname = '$nick' WHERE id_usuario = '$id' ";
```

Tras esto, los datos de usuario introducidos pasarían a ser las nuevas variables de sesión referentes a la información de usuario.

El siguiente apartado sería el de *CAMBIO DE CONTRASEÑA*, para el cual por un lado se introduce la contraseña actual y se repite para validar que sean iguales, por otro lado, se realiza lo

mismo con la contraseña nueva, y se realiza la siguiente consulta tras pulsar el botón del formulario:

```
$sql2 = "UPDATE usuarios SET password = '$md5passnew' WHERE id_usuario
='$id'";
```

Por último, se encuentra la sección *ELIMINAR CUENTA*, en la que se elimina el registro de la tabla de *usuarios* si y solo si este no cuenta con ningún servicio activo indicado en la tabla *detallescontrato*.

```
//Obtención de los servicios activos del usuario
$sql2 = "SELECT id_contrato FROM detallescontrato WHERE id_usuario= '$id'";
$servicios = mysqli_query($conexion, $sql2); //Ejecución de la consulta
$cuenta = mysqli_num_rows($servicios); //Recuento de los registros devueltos
        if ($cuenta==0){
                $sql3 = "DELETE FROM usuarios WHERE id_usuario = '$id'";
                ...
        }
```

- **Solicitar el Alta/Baja de Servicios IoT**

La Figura 31 muestra los servicios que aún no han sido solicitados por el usuario, para lograr representarlos, se han seguido los siguientes pasos:

- Primero se ha contado con un array compuesto por los nombres de todos los servicios ofrecidos por la plataforma. Por otro lado, se contaba con los servicios con los que cuenta el usuario inicialmente, así como el recuento de estos.
- Seguidamente y gracias a un *switch-case* se ha ido evaluado para cada caso de 0 a 12 servicios activos, eliminando (del array que contenía los 12 servicios) con la función *unset( )* de PHP, los que ya dispone el usuario.
- Finalmente se representan en una celda de la tabla que se ha creado para su visualización cada uno de los servicios que han quedado en el array gracias al uso de un *foreach( )*.

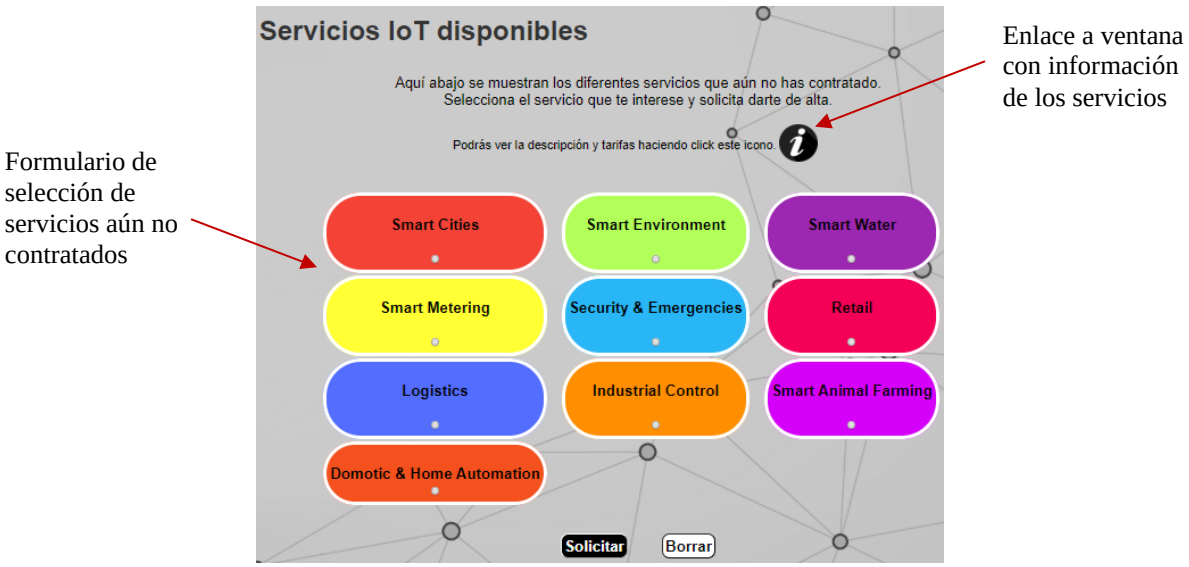


Figura 31. Solicitud de alta en servicios IoT

Al tratarse de un formulario en el que el usuario elige el servicio IoT, la consulta realizada en el fichero *solicitud.php*, para almacenar esta información en la base de datos sería la siguiente:

```
//Primero se obtiene el id_servicio ya que el formulario devuelve el nombre
$sql1 = "SELECT id_servicio FROM servicios WHERE nombre='$servicio'";
//La siguiente consulta corresponde a la creación de un nuevo registro
$sql2 = "INSERT INTO detallescontrato (id_contrato, id_servicio,id_usuario)
VALUES (null,'$id_servicio[0]','$id_usuario')";
```

Por otro lado, en el enlace *Solicitar Baja de Servicios IoT* del menú desplegable, se ofrece la posibilidad de cancelar los servicios con los que cuenta el usuario, el procedimiento sería muy similar al anterior visto para solicitar el alta, pero en este caso se eliminarán los registros correspondientes de la tabla *detallescontrato* con la siguiente consulta:

```
//Primero se obtiene el id_servicio ya que el formulario devuelve el nombre
$sql1 = "SELECT id_servicio FROM servicios WHERE nombre='$servicio'";
//La siguiente consulta corresponde a la eliminación del registro
correspondiente
$sql2 = "DELETE FROM detallescontrato WHERE id_servicio='$id_servicio[0]' AND
id_usuario='$id_usuario'";
```

Como se observa en la Figura 32 solo se muestran los servicios con los que cuenta el usuario en este momento, siendo su representación más fácil que en el caso de la solicitud ya que estos valores se disponen de la variable de sesión correspondiente.

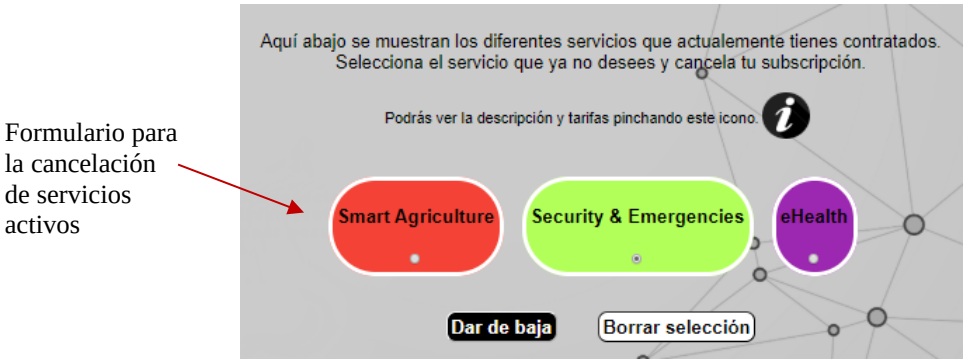


Figura 32. Solicitud de baja en servicios IoT

- **Visualización de Datos de los Servicios IoT**

Finalmente se encuentra el motivo principal que ha llevado a realizar esta aplicación, es decir, la visualización de los datos IoT. Dicha representación de los datos se ha centrado especialmente en el servicio de *Smart Agriculture* ya que se dispone de un sensor de temperatura y humedad, aunque sería aplicable a otros servicios que también requieran este tipo de medidas.

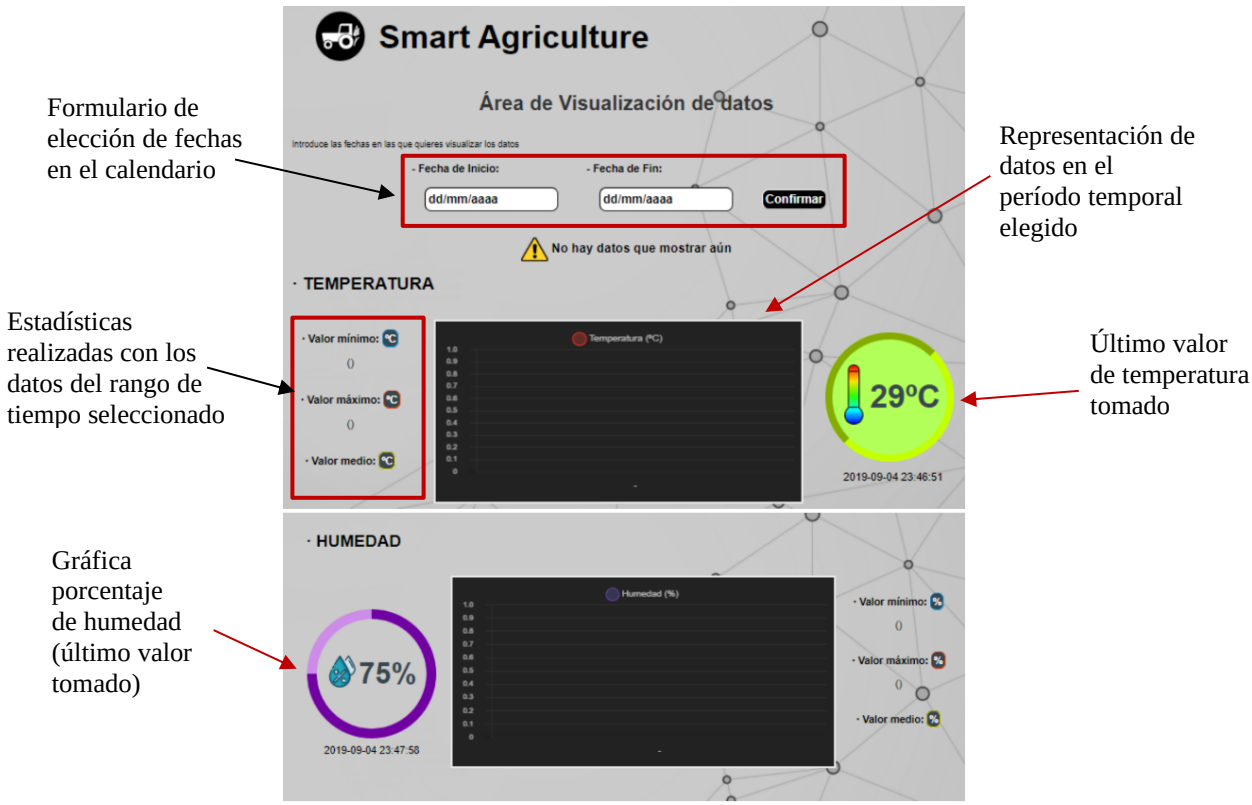


Figura 33. Pantalla de visualización de datos IoT

En la figura anterior se encuentra el área de visualización de datos mostrada al usuario, donde se puede elegir el rango de fechas en el que quiere visualizar las gráficas y estadísticas de ambos parámetros. Para ello se ha puesto a su disposición un formulario con entradas de tipo *date* lo que hace que se muestre un calendario facilitando la introducción de las fechas requeridas. Una vez introducidas se recarga esta misma página mostrando los datos en las gráficas y las estadísticas relacionadas.

Para la obtención de estos datos, se ha tomado de ejemplo la temperatura, ya que para la humedad solo cambiaría este parámetro, ejecutándose por tanto prácticamente la misma consulta. Además, estos datos están acotados por las fechas introducidas por el usuario para la visualización:

```
//Obtención de valores de temperatura y humedad para la gráfica en el tiempo
$sql2 = "SELECT medidas.temperatura, medidas.humedad, medidas.timestamp FROM
medidas INNER JOIN sensores ON medidas.id_sensor = sensores.id_sensor INNER
JOIN detallessensor ON sensores.id_sensor= detallessensor.id_sensor INNER JOIN
usuarios ON detallessensor.id_usuario = usuarios.id_usuario WHERE
usuarios.id_usuario= '$id_usuario' AND medidas.timestamp BETWEEN '$inicio' AND
'$finn'";

//Obtención del valor máximo de temperatura en el rango seleccionado
$sql3="SELECT medidas.temperatura, medidas.timestamp FROM medidas INNER JOIN
sensores ON medidas.id_sensor = sensores.id_sensor INNER JOIN detallessensor
ON sensores.id_sensor= detallessensor.id_sensor INNER JOIN usuarios ON
detallessensor.id_usuario = usuarios.id_usuario WHERE usuarios.id_usuario=
'$id_usuario' AND medidas.timestamp BETWEEN '$inicio' AND '$finn' ORDER BY
medidas.temperatura DESC LIMIT 1 ";
```

```
//Obtención del valor mínimo de temperatura en el rango seleccionado
$sql4="SELECT medidas.temperatura, medidas.timestamp FROM medidas INNER JOIN
sensores ON medidas.id_sensor = sensores.id_sensor INNER JOIN detallessensor
ON sensores.id_sensor= detallessensor.id_sensor INNER JOIN usuarios ON
detallessensor.id_usuario = usuarios.id_usuario WHERE usuarios.id_usuario=
'$id_usuario' AND medidas.timestamp BETWEEN '$inicio' AND '$finn' ORDER BY
medidas.temperatura ASC LIMIT 1 ";

//Obtención del valor medio de temperatura en el rango seleccionado
$sql5="SELECT AVG(medidas.temperatura) FROM medidas INNER JOIN sensores ON
medidas.id_sensor = sensores.id_sensor INNER JOIN detallessensor ON
sensores.id_sensor= detallessensor.id_sensor INNER JOIN usuarios ON
detallessensor.id_usuario = usuarios.id_usuario WHERE usuarios.id_usuario=
'$id_usuario' AND medidas.timestamp BETWEEN '$inicio' AND '$finn'";
```

La primera consulta hace referencia a los datos necesarios para la representación de las gráficas de temperatura y humedad (modificando la consulta como se ha dicho), las cuales se han conseguido gracias a la librería de Chart.js como se ha indicado anteriormente. En el apartado C del Anexo IV se adjunta el código en JavaScript que ha sido necesario incluir para la representación de estas gráficas.

Por otro lado, tanto para la temperatura como para la humedad, se ha querido mostrar un pequeño gráfico con el último valor recogido y el momento en el que se ha efectuado dicha medida, obteniendo dicho valor con la siguiente consulta:

```
$sql1 = "SELECT medidas.temperatura, medidas.humedad, medidas.timestamp FROM
medidas INNER JOIN sensores ON medidas.id_sensor = sensores.id_sensor INNER
JOIN detallessensor ON sensores.id_sensor= detallessensor.id_sensor INNER JOIN
usuarios ON detallessensor.id_usuario = usuarios.id_usuario WHERE
usuarios.id_usuario= '$id_usuario' ORDER BY medidas.id_medida DESC LIMIT 1 ";
```

Para el caso de la temperatura, se ha optado por situar este valor entre las etiquetas **, agregándole un icono y estilo deseado con HTML y CSS. En cambio, para el caso de la humedad, como se explica en la Tabla 4, se hace uso de una librería JavaScript y se ha desarrollado un código que la implementa, obteniendo un gráfico tipo "*doughnut*" dinámico que representa el porcentaje del total que indica el valor. Este código se ha adjuntado en el apartado D del Anexo IV.

Una vez terminada la sección de visualización de datos, se van a comentar una serie de aspectos comunes.

En varios de los bloques descritos con anterioridad, se ha utilizado una función escrita en JavasScript que permite actualizar ciertas zonas de la página sin tener que ser recargada manualmente por el usuario, la cual necesita de la librería *jquery.min.js* como se indica en la Tabla 4. Estas áreas son, por ejemplo: el menú desplegable (para actualizar los servicios activos en caso de solicitud o baja de alguno), los últimos valores de humedad y temperatura o los valores mostrados en la página de perfil de usuario. Una muestra de este código de recarga se muestra en el apartado E del Anexo IV.

Además, hay que mencionar que para todas las pantallas que incluyen formularios, en caso de no poder establecer la conexión que realiza las consultas a la base de datos o de no cumplirse las diferentes condiciones para cada formulario (por ejemplo: coincidencia de contraseñas, introducción de un nombre de usuario válido...), se han creado pantallas de error, similares a las

de validación que se verán en el siguiente apartado, que indican al usuario que ha habido un fallo de ejecución o bien error en los parámetros introducidos en los campos del formulario.

Por último, hay que indicar que todos los archivos que conforman la aplicación web se encuentran en la ruta: *Equipo → var → www → html*, ya que este es el directorio raíz del servidor.

3.1.3.2. Cliente MQTT

Tal y como se ha mencionado al inicio de este apartado del documento, el nodo servidor del sistema cuenta con dos funciones, y es que no se trata de un simple servidor web que muestra cualquier información almacenada en una base de datos a los usuarios. Estos datos son los recogidos por el sensor de temperatura y humedad en tiempo real. Para poder disponer de ellos, por tanto, este nodo debe actuar como un Cliente MQTT que se subscriba al tema en el que se publican.

Al igual que en el cliente MQTT implementado en la Raspberry, se va a hacer uso de Mosquitto, utilizando los mismos comandos para su instalación. Sin embargo, ahora el script a ejecutar cambia, puesto que como se ha descrito en el párrafo anterior este cliente es el que se subscribe al *topic* en el que se envían los datos de interés. El código de este cliente se puede encontrar en el Anexo II del documento.

Como cliente también podría publicar y mantenerse una comunicación bidireccional entre clientes, no obstante, en ese caso se ha simplificado para que los datos se envíen en una sola dirección.

3.1.4. Usuarios finales

Como muestra la Figura 14, al final del sistema IoT se encuentra el usuario final. Pese a que este no ha sido implementado como tal, cabe mencionarlo, ya que se trataría del destinatario de los servicios de IoT que se pretenden ofrecer con esta plataforma.

Dicho esto, cualquier usuario del servicio que disponga de un terminal con acceso a Internet podría acceder a la aplicación web que se pone a disposición, registrándose o iniciando sesión, solicitando el alta o baja de servicios de IoT y lo más importante, visualizando la información de interés de cada servicio de manera sencilla.

Tal y como se ha implementado esta plataforma, no es necesario que el usuario cuente con ningún conocimiento previo, puesto que todo el proceso que hay detrás es transparente para él. Esto facilita su utilización y permite contar con un mercado más amplio de clientes que utilizarían este sistema en caso de ser comercial.

4. Validación de la Plataforma en la Nube Privada para servicios de IoT

Para la validación de esta plataforma en la nube para servicios de IoT se ha procedido a la puesta en marcha del sistema al completo, con el fin de verificar el correcto funcionamiento de cada uno de los elementos de la arquitectura del sistema.

El primer paso ha consistido en hacer que el nodo cliente se encuentre activo, con el fin de que el sensor comience a realizar las medidas correspondientes y el cliente MQTT publique estos datos.

Para ello simplemente ha sido necesario mantener la Raspberry conectada a la fuente de alimentación y a Internet mediante un cable de Ethernet. Se ha ejecutado el código del fichero *publicadorSimple.py* que hace referencia al cliente MQTT que publica mensaje. En la Figura 34 se refleja dicha ejecución desde la línea de comandos de Raspbian, situándose primeramente en el directorio en el que se encuentra guardado el archivo.

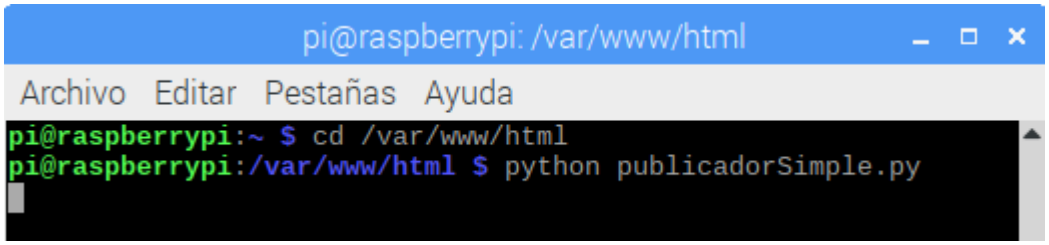


Figura 34. Ejecución del script del cliente MQTT publicador

Como se puede ver en el código correspondiente (Anexo II), este fichero se ejecuta en un bucle infinito, lo que quiere decir que los datos serán recogidos por el sensor y publicados hasta que se decida parar la ejecución de este programa. Además, se podría programar un ciclo de trabajo del sensor en función del servicio IoT que se quiera implementar.

Cabe mencionar que, para realizar las pruebas, el sensor se ha colocado en una planta situada en un ambiente doméstico interior tal y como se aprecia en la Figura 35.

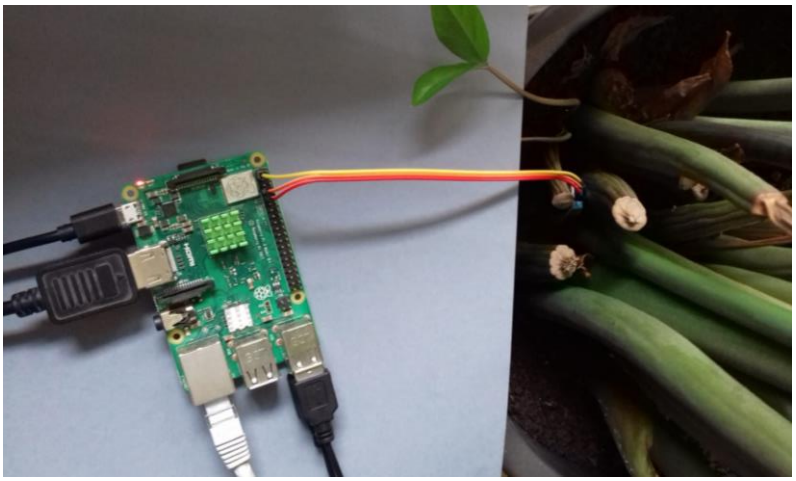


Figura 35. Montaje del nodo cliente

Respecto al broker utilizado, ya que no ha sido propiamente implementado, no se debe realizar ninguna acción específica para que esté activo. Es importante recordar que se ha usado un broker público, por lo tanto, no se ofrece garantía de funcionamiento sin interrupciones, sin embargo, sí que se puede esperar que por lo general esté activo y sea estable. Este será un factor que se deberá tener en cuenta una vez que el sistema completo esté funcionando.

A continuación, se pasa al nodo servidor, que al igual que el nodo cliente del sistema, debe permanecer activo de manera continua. Esto se debe a que por un lado ha de recoger los datos

enviados y almacenarlos y por otro debe ofrecer acceso a los usuarios que deseen utilizar la aplicación web en cualquier momento.

En este punto se consigue comprobar el funcionamiento del protocolo de comunicación implementado, verificando por tanto que el sensor recoge los datos correctamente, el cliente MQTT los publica con éxito y el broker utilizado los encamina hasta el cliente subscrito de manera satisfactoria.

De forma análoga al caso de la Raspberry, para iniciar el cliente MQTT subscrito se ha ejecutado el fichero *subscriptorSQL.py* en la Terminal de Debian.

Figura 36. Recepción de mensajes en el cliente MQTT subscrito

Observando la Figura 36, se ha podido concluir que tanto sensor, clientes MQTT y broker funcionan con normalidad, puesto que se puede ver como los datos enviados se muestran en la pantalla del cliente que los recibe. El siguiente paso ha sido confirmar que el código ejecutado no solo es capaz de recibir los datos y mostrarlos en la pantalla de la Terminal, sino que también los almacena en la base de datos de la forma indicada.

Para ello, se ha accedido a phpMyAdmin desde el navegador, visualizando así la base de datos, más concretamente la tabla de *medidas*, tal y como se ve en la Figura 37. En este sentido, puede confirmarse que el código del cliente MQTT alojado en el nodo servidor cumple todas las funciones requeridas.

Figura 37. Tabla *medidas* (almacenamiento de datos recogidos por el sensor)

Por último, tras ver que el proceso de recogida de datos, el protocolo de comunicación y el almacenamiento se llevan a cabo satisfactoriamente en este sistema, quedaría validar la aplicación web y sus funcionalidades desde el punto de vista de un usuario final.

Por ello no solo se ha comprobado que la página web sea accesible desde cualquier terminal fuera de la red del servidor, sino también su correcta visualización, prestando especial interés a los

puntos en los que el usuario interactúa con la aplicación para visualizar, modificar o crear registros en la base de datos como se muestra en los siguientes apartados:

- **Registro de Usuarios**

Para verificar el correcto registro de usuarios se ha decidido crear uno desde la aplicación, como se puede ver en la Figura 38, una vez se han introducido los datos se redirige a una página en la que se indica que el registro se ha realizado con éxito. Además, se ha comprobado también la creación de este nuevo usuario en la base de datos, como se refleja en la Figura 39.

Figura 38. Proceso de registro de usuarios

id_usuario	nombre	email	telefono	nickname	password
1	Usuario Ejemplo	usuario@example.com	600000006	user	ee11cbb19052e40b07aac0ca060c23ee

Figura 39. Tabla *usuarios* tras el registro de un cliente

- **Inicio de Sesión**

El siguiente paso ha sido verificar que este usuario puede iniciar sesión en la aplicación sin problemas, como se ve en la Figura 40, al completar el formulario se redirige al perfil de este usuario mostrando su información, lo que quiere decir que las consultas en la base de datos y el establecimiento de una sesión se llevan a cabo correctamente.

Figura 40. Inicio de sesión

- **Modificar Datos de Usuario y Eliminar cuenta**

Una vez dentro de la aplicación, una de las opciones que se le ofrece al usuario es la de modificar sus **datos personales**, tras cambiarlos, se envía al usuario a la página donde aparece un mensaje de confirmación del cambio.

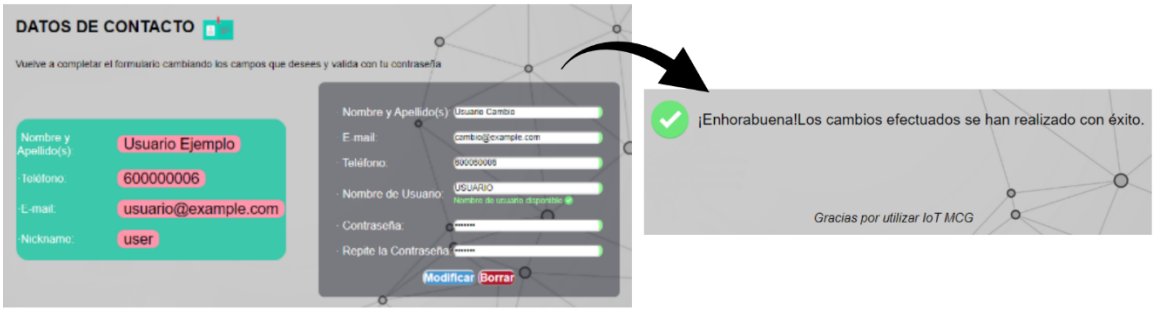


Figura 41. Modificación de datos personales del usuario

Al acceder a la base de datos se ve que efectivamente, los datos han sido modificados, tratándose del mismo usuario, ya que el campo de *id_usuario* sigue siendo el mismo.

id_usuario	nombre	email	telefono	nickname	password
1	Usuario Cambio	cambio@example.com	600060006	USUARIO	ee11cbb19052e40b07aac0ca060c23ee

Figura 42. Tabla *usuarios* (se aprecian los cambios realizados)

Al igual que en el caso de la información personal, si se rellena el formulario de cambio de **contraseña** correctamente, se muestra al usuario que la acción se ha realizado con éxito.

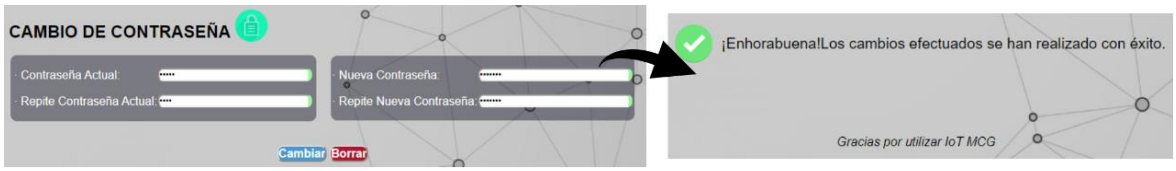


Figura 43. Cambio de contraseña

Ya que la contraseña se guarda cifrada en la base de datos, no se puede apreciar cual ha sido el cambio, pero en la Figura 44 se puede apreciar como la cadena resultante del cifrado ha cambiado.

id_usuario	nombre	email	telefono	nickname	password
1	Usuario Ejemplo	usuario@example.com	600000006	user	ee11cbb19052e40b07aac0ca060c23ee

id_usuario	nombre	email	telefono	nickname	password
1	Usuario Ejemplo	usuario@example.com	600000006	user	f8032d5cae3de20fcec887f395ec9a6a

Figura 44. Tabla *usuarios* tras cambio de contraseña (diferente cifrado)

Por último, se ha procedido a eliminar la cuenta creada, apreciándose también un resultado exitoso. Se puede observar en la Figura 46 como la tabla de *usuarios* de la base de datos ya no cuenta con ningún registro.

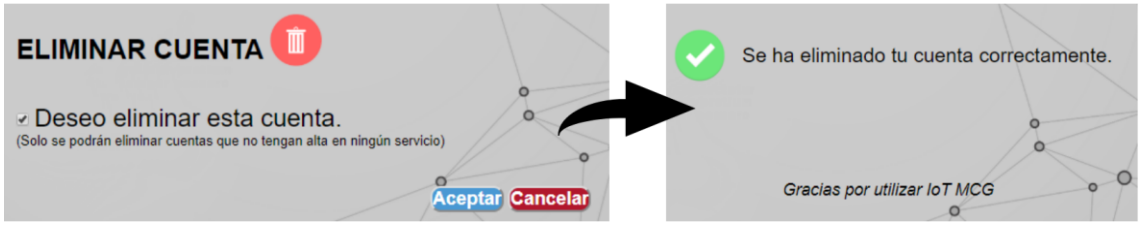


Figura 45. Borrado de cuenta de usuario

id_usuario	nombre	email	telefono	nickname	password

Figura 46. Tabla *usuarios* (tras eliminar la cuenta)

- **Solicitar Alta/Baja en Servicios IoT**

Seguidamente se ha verificado el funcionamiento de las opciones de solicitar el alta o dar de baja los servicios IoT que se ofrecen en la aplicación. La Figura 47 muestra la solicitud de un par de servicios por parte del usuario creado anteriormente.

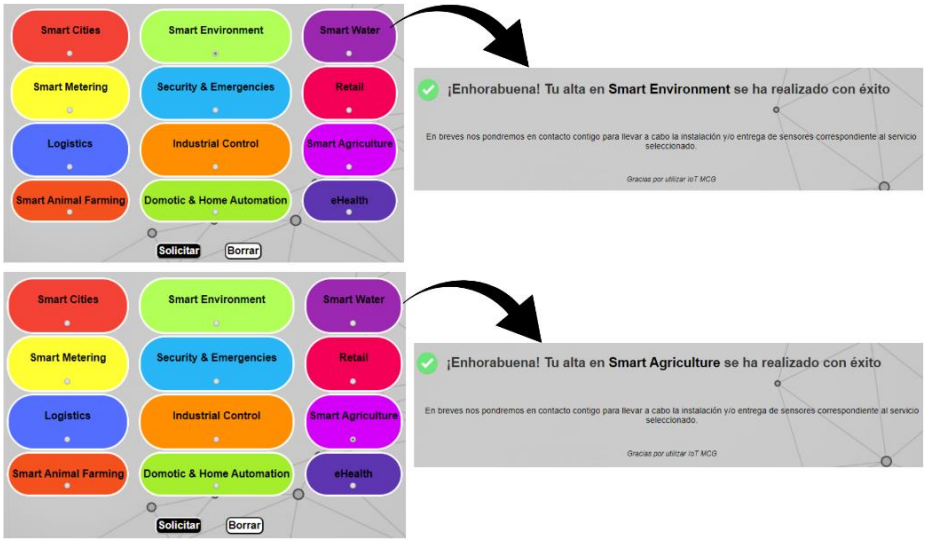


Figura 47. Solicitud de alta en diferentes servicios de IoT

Al acceder a la base de datos, se puede ver que en la tabla *detallescontrato* se han creado estos registros de forma correcta.

id_contrato	id_servicio	id_usuario
53	2	1
54	9	1

Figura 48. Registros creados en la tabla *detallescontrato*

- **Visualización de datos de los servicios de IoT**

Una vez visto que las opciones anteriores se ejecutan correctamente, se ha procedido a la comprobación del objetivo principal de esta aplicación, es decir, que el usuario pueda visualizar los datos recogidos por los sensores. Aunque para la implementación de este sistema se ha hecho uso de un único sensor de temperatura y humedad centrado para *Smart Agriculture*, la aplicación ha sido diseñada para que el usuario pueda consultar diferentes servicios de IoT, y, por tanto, visualizar diferentes sensores, es decir, la plataforma en la nube es flexible, modular y escalable.

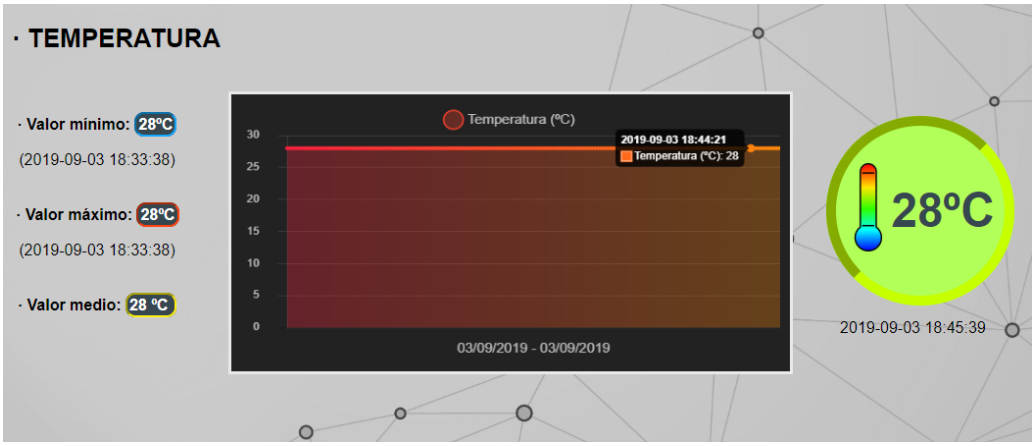


Figura 49. Visualización de datos relativos a temperatura según fechas indicadas por el usuario

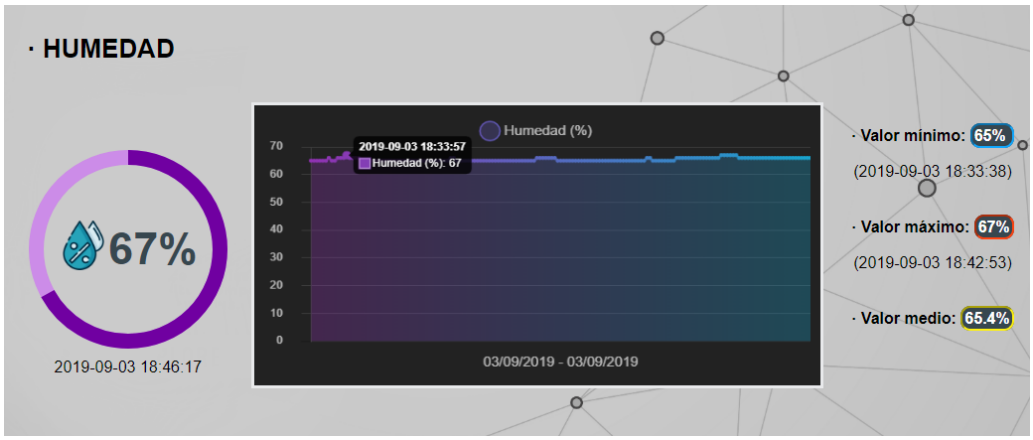


Figura 50. Visualización de datos relativos a humedad según las fechas indicadas por el usuario

La Figura 489y la Figura 50 muestran la pantalla mostrada al usuario una vez que este accede al área de visualización de datos y selecciona las fechas en las que quiere ver la información. Como se puede observar, las gráficas se representan correctamente, indicando en cada punto representado, el valor y el momento exacto de medida si se sitúa el cursor por encima y representando únicamente los valores incluidos en el rango seleccionado (pudiendo tratarse de diferentes períodos temporales: año, mes, semanas, días...). Además, se observa que aparecen las estadísticas realizadas para las medidas en las fechas seleccionadas como se había programado para cada uno de los casos. Por otra parte, independientemente de las fechas escogidas por el usuario para la representación de los datos, los indicadores tanto de temperatura como de humedad muestran el último valor recogido por los sensores, actualizándose sin que haya que recargar el navegador, tal y como se deseaba.

Después de esto, se puede decir que la solución implementada cumple correctamente todas las funciones requeridas.

5. Conclusiones y líneas de trabajo futuras

5.1. Conclusiones

Actualmente, Internet de las Cosas es un tema relevante de investigación tanto para la industria como para la academia e institutos de investigación. IoT triplicará entre 2018 y 2015 el número de objetos y/o dispositivos conectados a Internet, alcanzando hasta 25 mil millones de dispositivos, con unos ingresos que se cuadriplicarán a $1.1 billones de dólares. La combinación de IoT con la computación en la nube o Cloud Computing permite el almacenamiento, procesado y acceso de los datos, así como su monitorización y análisis para ofrecer servicios a todo tipo de aplicaciones y consiguiendo minimizar los problemas de memoria y espacio que se presentan al manejar tal cantidad de información. Esto supone una verdadera revolución en la industria debido a la automatización de tareas que mejoren los procesos de ejecución, tiempo de producción y costes. Por otro lado, puede llevar al surgimiento de nuevos modelos de negocio (a nivel local y global) en ámbitos tan dispares como, por ejemplo; educación, comunicaciones, ciencia...

En este Trabajo Final de Grado se ha diseñado, implementado y evaluado una plataforma en la nube privada de bajo coste y con herramientas de código abierto para servicios de IoT, la cual permite el acceso ubicuo de los datos en tiempo real, modularidad, flexibilidad, fiabilidad, y escalabilidad para diferentes servicios IoT.

Se propuso una arquitectura de cuatro capas para el sistema IoT, la capa de percepción está formada por el sensor DHT11 y una placa de desarrollo Raspberry Pi 3 B+. El sensor permite monitorizar diferentes parámetros, tales como, humedad y temperatura. La Raspberry Pi 3 B+ se comunica con el servidor mediante el protocolo de código abierto MQTT, el cual está basado en la pila de protocolos TCP/IP. Para llevar a cabo el envío de los datos entre los clientes, se ha hecho uso del broker público, *test.mosquitto.org*, el cual permite la encriptación de los datos a través del puerto 8883, y las versiones TLS v1.3, v1.2 y v1.1 con certificado x509. La capa de almacenamiento, procesado y visualización de los datos de usuario ha sido implementada mediante un servidor, y una base de datos estructurada y escalable. El servidor consta de una aplicación web donde se visualizarán los servicios y datos de los clientes IoT, se ha utilizado HTML junto CSS para definir la estructura y diseño de las diferentes pantallas que lo conforman, integrando fácilmente PHP para conseguir así una página web dinámica. Además, para lograr ciertos aspectos de la página web, como puede ser actualizar determinadas partes sin que el usuario tenga que recargar la página o contar con diferentes representaciones de los datos en tiempo real que ofrecen los servicios de IoT a los que se subscribe el usuario, se ha hecho uso de algunas librerías de código abierto de JavaScript. Como sistema de gestión de la base de datos se usó MySQL, la cual es una de las bases de datos de código abierto más populares en entornos de desarrollo web. Por último, la cuarta capa de la arquitectura del sistema IoT es el usuario final, el cual podrá visualizar los datos en tiempo real de los servicios IoT que tenga contratados.

Para la validación de la plataforma en la nube, se realizaron medidas en tiempo real en una aplicación de *Smart Agriculture* IoT donde se monitorizaron parámetros como la humedad y temperatura de una planta. En este sentido, se verificó el correcto funcionamiento de cada uno de los elementos de la arquitectura del sistema IoT operando tanto de forma individual como conjunta de la plataforma. En la aplicación web, se comprobaron correctamente todas las funcionalidades, tales como, registro de usuarios, inicio de sesión, modificación de datos, alta y

baja de servicios de IoT, consulta de datos en tiempo real mediante la visualización en gráficas y estadísticas (pudiendo tratarse de diferentes períodos temporales: años, mes, semanas, días...).

5.2. Líneas de trabajo futuras

Como se ha podido comprobar, todos los puntos del sistema desarrollado en este proyecto han ofrecido el resultado que se esperaba de ellos, por lo que se puede decir que se trata de plataforma válida para su uso en servicios de IoT. Seguidamente se exponen una serie de líneas de trabajo, considerando la solución implementada como base:

- Se sugiere realizar una comparativa de prestaciones con otras plataformas, tomando como posibles indicadores: la escalabilidad, fiabilidad, seguridad, capacidad de conexiones simultáneas, tanto de usuarios como de mensajes.
- El nodo servidor como se ha visto, se ha implementado en un ordenador portátil, lo que ofrece algunas desventajas, ya que se necesita que se encuentre siempre activo para la recepción de información y su almacenamiento en la base de datos. Por ello se recomienda hacer la implementación de este nodo en un dispositivo con características más coherentes con las tareas que se llevan a cabo en dicho nodo.
- Trabajar más en profundidad el aumento de seguridad de la plataforma, en puntos críticos como pueden ser: el broker MQTT (ya que se ha utilizado uno público) o el nodo servidor del sistema, ya que en él se almacenan gran cantidad de datos.
- Implementar una aplicación en Android, con las mismas prestaciones que se ofrecen al usuario en la aplicación web que se ha desarrollado. Esto presentaría como ventajas: la visualización de la aplicación adaptada a terminales con este sistema operativo y una mayor comodidad en el uso de esta plataforma para el usuario.

Bibliografía

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari and M. Ayyash. "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications" *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347-2376, Fourthquarter 2015.

[2] Pallavi Sethi and Smruti R. Sarangi, "Internet of Things: Architectures, Protocols, and Applications," Journal of Electrical and Computer Engineering, vol. 2017, Article ID 9324035, 25 pages, 2017. https://doi.org/10.1155/2017/9324035

[3] Anirudh V. K. "What Is Cloud Computing Architecture: Front-End & Back-End Explained", 2019. https://it.toolbox.com/tech-101/what-is-cloud-computing-architecture-front-end-back-end-explained

[4] Luis Joyanes Aguilar. "Computación en la nube. Notas para una estrategia española en *Cloud Computing*", 2009. http://revista.ieee.es/article/view/406/706

[5] Peter Mell, Timothy Grance. "The NIST Definition of Cloud Computing", 2011. http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf

[6] "Open Source IoT Platform | Top 10 Open Source IoT Platform", 2018. http://www.thetips4you.com/open-source-iot-platform-top-10-open-source-iot-platform/

[7] Kaa IoT Platform. https://www.kaaproject.org/

[8] OpenIoT Documentation. https://github.com/OpenIotOrg/openiot/wiki/Documentation

[9] About ThingSpeak. https://thingspeak.com/pages/learn_more

[10] AWS IoT: Guía del desarrollador. https://docs.aws.amazon.com/es_es/iot/latest/developerguide/iot-dg.pdf#what-is-aws-iot

[11] "Internet of Things (IoT) technologies and solutions: PaaS and SaaS", 2018. https://docs.microsoft.com/en-us/azure/iot-fundamentals/iot-services-and-technologies

[12] "Azure IoT reference architecture", 2019. https://docs.microsoft.com/en-us/azure/architecture/reference-architectures/iot/

[13] Google Cloud IoT. https://cloud.google.com/solutions/iot/?hl=es

[14] DHT11 Humidity & Temperature Sensor. https://www.mouser.com/ds/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf

[15] Raspberry Documentation. https://www.raspberrypi.org/documentation/usage/gpio/

[16] Michael Yuan. "Conociendo MQTT", 2018. https://www.ibm.com/developerworks/ssa/library/iot-mqtt-why-good-for-iot/index.html

[17] Roger Light. "MQTT man page". https://mosquitto.org/man/mqtt-7.html

[18] GitHub Public Brokers. https://github.com/mqtt/mqtt.github.io/wiki/public_brokers

[19] Comparison of the usage of Apache vs. Nginx vs. Lighttpd for websites. https://w3techs.com/technologies/comparison/ws-apache,ws-lighttpd,ws-nginx

[20] Historical trends in the usage of web servers. https://w3techs.com/technologies/history_overview/web_server

[21] Robin Muilwijk. “Top 5 open source web servers”, 2016. https://opensource.com/business/16/8/top-5-open-source-web-servers

[22] Guido van Rossum. “El tutorial de Python”, 2009. http://docs.python.org.ar/tutorial/pdfs/TutorialPython2.pdf .

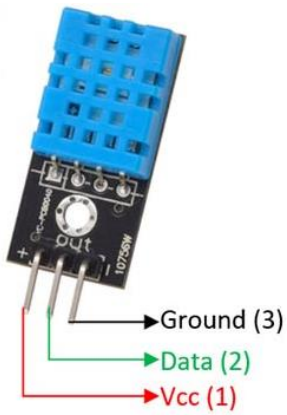
[23] Chart.js Documentation. https://www.chartjs.org/docs/latest/

[24] Manual de PHP. https://www.php.net/manual/es/index.php

[25] JavaScript Tutorial. https://www.w3schools.com/js/

Anexos

Anexo I: Datos Técnicos del Sensor de Temperatura y Humedad DHT11



Operating Voltage	DC	3.3V - 5.5V
Current Suply	Measuring	0.5mA - 2.5mA
	Average	0.2mA - 1mA
	Standby	100uA - 150uA
Output	Serial data	
Temperature Range	0ºC -50ºC	
Humidity Range	20% - 90% RH	
Accuracy	±2ºC and ±5% RH	
Sampling Rate	1Hz (reading every second)	

Tabla 5. Ficha de datos del sensor DHT11

Anexo II: Recogida de Datos y Publicación de Cliente MQTT

```
while True:

    import paho.mqtt.client as mqtt
    import time
    import ssl
    import Adafruit_DHT

    sensor=Adafruit_DHT.DHT11 # Tipo de sensor (DHT11, DHT22 o AM2302)
    gpio=4 #Indica a qué pin GPIO se conectan los datos del DHT11
    humidity, temperature = Adafruit_DHT.read_retry(sensor, gpio) #Lectura

    broker_address = "test.mosquitto.org" #Dirección del broker
    broker_port = 8883 #Puerto usado por el broker
    topic = "iot/agriculture" #Tema en el que se publican los mensajes

    idsensor=1
    date=time.strftime("%Y-%m-%d %X")
    medida = str(idsensor) + "/" + str(temperature) + "/" + str(humidity) +
"/" + date

    client= mqtt.Client('Publicador')
    client.tls_set("/var/www/html/mosquitto.org.crt",
tls_version=ssl.PROTOCOL_TLSv1_2)
    client.tls_insecure_set(True)
    client.connect(broker_address, broker_port, 60)

    if humidity is not None and temperature is not None:
```

```
        client.publish(topic, medida) #Publicación de medida correcta
    else:
        medida = str(idsensor) + "/" + "ERROR" + "/" + "ERROR" + "/" + date
        client.publish(topic, medida) #Publicación de medida errónea
```

Anexo III: Subscripción de Cliente MQTT y Almacenamiento en Base de Datos

```
#!/usr/bin/python
import paho.mqtt.client as mqtt
import time
import ssl
import MySQLdb

db = MySQLdb.connect(host="localhost", user="root", passwd="Iot.Mysql",
db="iotservices")

cur = db.cursor()

broker_address = "test.mosquitto.org"
broker_port = 8883
topic = "iot/agriculture"

def on_connect(client, userdata, flags, rc):
        print("Connect with result code" + str(rc))
        print("UserData= " + str(userdata))
        print("flags= " + str(flags))
        print("")
        client.subscribe(topic)

def on_message(client, userdata, message):

        m = str(message.payload.decode("utf-8"))

        mensaje = m.split('/')
        h=float(mensaje[2])

        if h<=100:
                cur.execute(''' insert into iotservices.medidas (id_medida,
id_sensor, temperatura, humedad, timestamp) values (null,'%s', '%s', '%s',
'%s')''' % (mensaje[0], mensaje[1], mensaje[2], mensaje[3]))

                db.commit()
                print("id_sensor=", mensaje[0])
                print("temperatura=", mensaje[1])
                print("humedad=", mensaje[2])
                print("timestamp=", mensaje[3])
                print("")
        else:
                print("Error en la medida")

client = mqtt.Client('Cliente', userdata="UsuarioPrueba")
```

```
client.on_connect = on_connect
client.on_message = on_message
client.tls_set("/var/www/html/mosquitto.org.crt",
tls_version=ssl.PROTOCOL_TLSv1_2) //Certificado para protocolo encriptado
client.tls_insecure_set(True)
client.connect(broker_address, broker_port, 60)
client.loop_forever()
```

Anexo IV: Fragmentos de código de la Aplicación Web

A. Galería de imágenes:

```
var slider=$('#slider'); //Almacena división #slider en una variable
var siguiente= $('#btn-next'); //Almacena botón derecho
var anterior= $('#btn-prev'); //Almacena botón izquierdo
//Introduce imagen de la última sección antes de la primera
$('#slider section:last').insertBefore('#slider section:first');

slider.css('margin-left', '-'+100+'%');
//Función que muestra imagen siguiente (la situada a la derecha)
function moveD(){
        slider.animate({
        marginLeft:'-'+200+'%'} , 700, function(){
                        $('#slider section:first').insertAfter('#slider
section:last')
                        slider.css('margin-left', '-'+100+'%');
        });
}
//Función que muestra imagen anterior (la situada a la izquierda)
function moveI(){
        slider.animate({
        marginLeft:0} , 700, function(){
                        $('#slider section:last').insertBefore('#slider
section:first')
                        slider.css('margin-left', '-'+100+'%');
        });
}
//Función muestra imagen siguiente automáticamente cada 5s
function auto(){
        intervalo = setInterval(function(){
                moveD();
        }, 5000);
}
//Muestra siguiente imagen al pulsar botón derecho
siguiente.on('click', function(){
        moveD();
        clearInterval(intervalo);
        auto();
});
//Muestra imagen anterior al pulsar botón izquierdo
```

```
anterior.on('click', function(){
        moveI();
        clearInterval(intervalo);
        auto();
});
//Ejecuta el deslizamiento automático
auto();
```

B. Comprobación de nombre de usuario en el formulario de registro:

```
<script type="text/javascript">

function comprobar(nick){
        var url = 'check_nick.php';
        var pars = ('nickname=' + document.getElementById('name').value);
        var myAjax = new Ajax.Updater('comprobar_mensaje', url, {method:
'get', parameters: pars});
}

</script>
```

C. Representación de gráficas:

Este sería el código correspondiente a la gráfica de humedad, el de temperatura sería prácticamente idéntico, cambiando solo los colores y parámetros introducidos.

```
<script type="text/javascript">

var ctx = document.getElementById("migrafico4").getContext('2d');
var fechas = '<?php echo $fechas; ?>';
var width = window.innerWidth || document.body.clientWidth;
var gradientStroke = ctx.createLinearGradient(0, 0, width, 0);
gradientStroke.addColorStop(0, "#9C27B0");
gradientStroke.addColorStop(0.5, "#00BCD4");
gradientStroke.addColorStop(1, "#69F0AE");
var gradientFill = ctx.createLinearGradient(0, 0, width, 0);
gradientFill.addColorStop(0, "rgba(156, 39, 176, 0.3)");
gradientFill.addColorStop(0.5, "rgba(0, 188, 212, 0.3)");
gradientFill.addColorStop(1, "rgba(105, 240, 174, 0.3)");
var myChart =new Chart(ctx,{
        type:'line',
        data:{
                labels: [<?php echo $timestamp; ?>],
                datasets: [{
                        label: 'Humedad (%)',
                        data: [<?php echo $humedad; ?>],
                        backgroundColor: gradientFill,
                        borderColor: gradientStroke,
                        pointBorderColor: gradientStroke,
                        pointBackgroundColor: gradientStroke,
                        pointHoverBackgroundColor: gradientStroke,
```

```
                        pointHoverBorderColor: gradientStroke,
                        borderWidth:0.5
                }]
        },
        options:{
                legend: {
                        labels:{
                                fontSize: 15,
                                usePointStyle: true,
                                fontColor: "rgba(255,255,255,0.7)"
                        }
                },
                scales: {
                        yAxes: [{
                                ticks: {
                                        padding: 20,
                                        fontColor: "rgba(255,255,255,0.7)",
                                        beginAtZero:true
                                },
                                gridLines: {
                                color:"rgba(127,127,127,0.3)"
                                }
                        }],
                        xAxes: [{
                                scaleLabel:{
                                        display:true,
                                        labelString: fechas,
                                        fontSize: 15,
                                        padding: 5,
                                        fontColor: "rgba(255,255,255,0.7)"
                                },
                                ticks: {
                                        display: false,
                                        padding: 20,
                                        fontColor: "rgba(255,255,255,0.7)"
                                },
                                gridLines: {
                                        color:"rgba(127,127,127,0.3)"
                                }
                        }]
                }
        }
});
</script>
```

D. Gráfica Circular para la representación de porcentaje:

```
<script>
        $(function(){
                $('.chart').easyPieChart({
                        size: 200,
                        barColor: '#7000a1',
```

```
                        backgroundColor: '#000',
                        scaleColor: false,
                        lineWidth: 15,
                        trackColor: '#cc8ce8',
                        lineCap: 'circle',
                        animate: 900,
                });
        });
</script>
```

E. Actualización de secciones de la página sin recargar el navegador:

```
<script type="text/javascript">
        $(document).ready(function(){
                setInterval(
                        function(){
                                $('#here').load('recarga.php');
                        },4000
                );
        });
</script>
```