Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Dra. Dª. *Macarena Espinilla Estévez*

Y Dr. Dº. *Miguel Damas Hermoso*, tutores

del Trabajo Fin de Grado titulado:

Diseño, construcción e implementación de un prototipo de producto IoT,

que presenta D. *Manuel Navas Damas*, autorizan

su presentación para defensa y evaluación en la

Escuela Politécnica Superior de Jaén.

Jaén, Junio de 2018

El alumno:

Manuel Navas Damas

Los tutores:

Macarena Espinilla Estévez

Miguel Damas Hermoso

Tabla de contenido

Tabla de Figuras

Capítulo I

Introducción

En este primer capítulo vamos a exponer los motivos que nos han llevado a escoger este trabajo, asi como la planificación para llevarlo a cabo incluyendo tanto los costes de tiempo como de presupuesto. Además terminaremos especificando cómo se va a estructurar esta memoria para poder ofrecer una visión global de todos los temas que vamos a abarcar a lo largo del trabajo.

1.1. Motivación

Hoy en día Internet es conocido por todas las sociedades modernas existen y es indudable que el impacto que ha supuesto en casi todos los ámbitos de la vida moderna es superior al de cualquier otra tecnología creada por la mano del hombre. Durante el breve periodo histórico que lleva con nosostros esta herramienta de ha revolucionado la forma de comunicarnos, lo que ha supuesto una mejora sustancial en diferentes ámbitos de la rutina diaria de cualquier persona. Sin embargo es ahora cuando Internet está dando un paso de gigante en su historia, gracias a los nuevos objetivos que propone el concepto del Internet de las Cosas (IoT). No en vano el IoT es considerado por muchos expertos como la primera gran evolución de Internet desde su creación, ya que es un concepto que a priori puede parecer simple pero esconde una gran complejidad. A medida que aumenta el número de dispositivos (Things) enviando todo tipo de datos a la red también proliferan las tecnologías que aprovecharán toda esta ingente cantidad de datos para aprender de ellos y generar conocimiento que nos dará la oportunidad de progresar como sociedad como nunca antes habia sucedido.

Por este motivo este concepto resulta tan interesante y hemos decidido centrar este trabajo en aprender más sobre todo lo que conlleva un desarrollo de un producto IoT, desde la concepción hasta la implementación final, lo que nos permitirá abarcar diferentes aspectos muy importantes que juntos proporcionan una buena visión global de la complejidad que el desarrollo de este tipo de proyectos supone.

1.2. Objetivos

Nuestro propósito es desarrollar un **prototipo de producto IoT** que nos permita monitorizar cualquier tipo de depósito para poder conocer en todo momento el nivel que presenta, poder seleccionar distintos tipos de depósitos y modificar tanto sus dimensiones como algunas preferencias. Además también deberá existir la posibilidad de establecer un umbral mínimo sobre el nivel del depósito, de forma que si el nivel disminuye de dicho umbral se le notifique al usuario de alguna forma. Todo esto acompañado de una **herramienta de gestión y visualización** que permita al cliente modificar parámetros y comprobar el estado de dicho depósito de una forma sencilla y desde cualquier lugar con acceso a Internet.

Teniendo en cuenta esta propuesta podemos definir algunos objetivos que cumplir durante la realización del trabajo:

- Estudio de los conceptos asociados a las tecnologías que se van a utilizar en el proyecto.
- Analizar las diferentes kits de desarrollo hardware existentes en el mercado para el desarrollo de prototipos IoT y seleccionar una apropiada para la elaboración del proyecto.
- Construir e implementar un dispositivo IoT mediante el kit de desarrollo hardware seleccionado y los elementos electrónicos (sensores, actuadores, etc.) adicionales necesarios para llevar a cabo la propuesta explicada anteriormente.

- Desarrollar dos alternativas para una herramienta de gestión y visualización que junto con el dispositivo IoT ofrezcan las funcionalidades establecidas para el proyecto. Una de ellas deberá ser una Plataforma IoT y la otra se realizará mediante el desarrollo desde cero de una aplicación para dispositivos móviles con sistema operativo Android.
- Estudiar las distintas Plataformas IoT existentes y escoger una de ellas para utilizarla en nuestro proyecto como herramienta de gestión y visualización.
- Solucionar la comunicación entre la herramienta de gestión y visualización y el dispositivo (para las dos alternativas), de manera que a través de dicha herramienta podamos acceder a los datos y la configuración del dispositivo en cualquier momento y desde cualquier lugar con acceso a Internet.
- Diseñar e imprimir en 3D una carcasa que aporte una apariencia agradable al prototipo y además permita poder anclarse fácilmente a los depósitos que podemos encontrar en los hogares.

1.3. Planificación del proyecto

En este apartado vamos a ver cómo hemos planificado la realización del proyecto, tanto desde el punto de vista del tiempo necesario para su realización como de los costes que se requerirán para poder llevar a cabo su desarrollo e implementación.

1.3.1. Planificación de tiempos

Para este propósito vamos a tener en cuenta una estimación optimista de la duración para cada una de las tareas que tendremos que abarcar a lo largo del desarrollo de este proyecto. Para ello hemos realizado la siguiente tabla que ilustra cada una de las tareas y el tiempo estimado para las mismas, dando como resultado un tiempo de aproximadamente 4 meses (992 horas a 8 horas por día) para poder llevar a cabo la propuesta del proyecto.

Tarea	Duración (Horas)
Búsqueda información sobre IoT	40
Búsqueda información sobre Tecnologías habilitadoras	40
Búsqueda información Plataformas IoT	40
Búsqueda de bibliografía adicional	64
Desarrollo del dispositivo IoT	168
Desarrollo herramienta de gestión (Plataforma IoT)	112
Desarrollo herramienta de gestión (Aplicación Android)	280
Testeo y corrección de errores	56
Redacción de la memoria	192
Total	**992**

- **Diagrama de Grant**

Un diagrama de Gantt es una representación gráfica de las tareas que constituyen un proyecto y su duración a lo largo de un periodo determinado de tiempo. Es una muy buena herramienta para controlar el seguimiento de las tareas y comprobar si estamos cumpliendo con el progreso estimado del proyecto, como se puede observar en la figura 1.

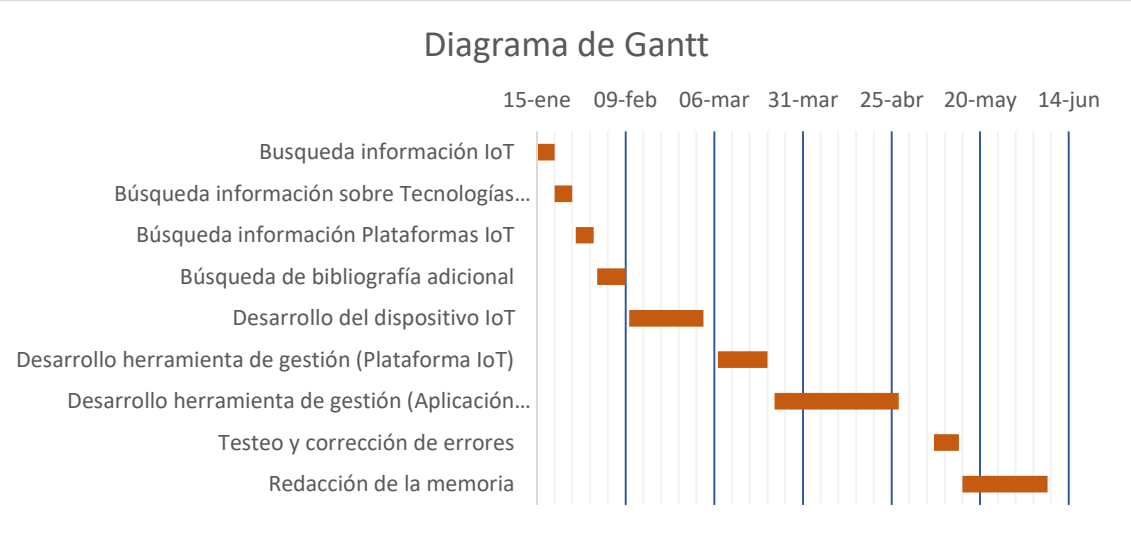


Figura 1: Diagrama de Gantt

1.3.2. Análisis de costes

Realizar un estudio sobre el presupuesto que vamos a necesitar para llevar a cabo un proyecto es una fase obligatoria en la etapa de planificación. Además en la mayor parte de los casos puede resultar una parte fundamental para poder llevar a cabo el proyecto, ya que si el coste total estimado es muy alto puede suponer la cancelación del proyecto. En el presupuesto se valoran económicamente los costes que se necesitan para llevar a cabo el desarrollo, y en el se incluyen tanto los costes de hardware y licencias de software como los costes de mano de obra de los desarrolladores.

Costes de la mano de obra			
	Sueldo mensual (€)	Meses	Coste total (€)
Análisis	1494,66	1	1494,66
Diseño	1494,66	1	1494,66
Implementación	1197,85	1	1197,85
Memoria	1197,85	1	1197,85
Coste bruto			**5385,02**
IVA (21%)			**1130,85**
Coste neto			6515,8742

Costes de Hardware			
	Coste unitario (€)	Unidades	Coste total (€)
NodeMCU	4,99	1	4,99
Sensor HC-SR04	2,99	1	2,99
Conectores	0,1	4	0,40
Impresión 3D	0,50	1	0,50
Coste bruto			**8,88**
IVA (21%)			**1,86**
Coste neto			10,74

Deberíamos también añadir los costes del software, sin embargo el único programa que podría requerir una suscripción de pago es la *Plataforma IoT* y como veremos más adelante solamente necesitamos utilizar una cuenta gratuita por lo que el coste de software para nuestro proyecto es cero.

Por tanto el **presupuesto final** para llevar a cabo nuestro proyecto es de 6525€ aproximadamente, pero debemos tener en cuenta que la **mayor parte de dicho**

presupuesto se destinaría a la mano de obra necesaria para su desarrollo, con lo cual si tenemos en cuenta únicamente el hardware necesario para construir el dispositivo *IoT* vemos que el **coste para llevar a cabo un producto IoT de este tipo es muy pequeño** (alrededor de los 10€). Además si se quisiera desarrollar en masa estos dispositivos con el objetivo de ofrecer un producto destinado a un proveedor de algún producto susceptible de almacenarse en un depósito se podría crear un modelo de negocio similar al de Dash Button de Amazon, ya que se **reduciría aún más el coste por unidad** (se podría conseguir un precio cercano a los 5€ por dispositivo) y sería muy asequible para estas empresas regalarlo a sus clientes para ofrecerles un servicio más cómodo.

1.4. Estructura de la memoria

Este apartado lo vamos a dedicar a describir la estructura del trabajo, de forma que el lector tenga una visión general de los contenidos que se verán en los sucesivos capítulos:

- **Capítulo I: Introducción**. En este capítulo se describen los motivos que nos han llevado a abarcar este proyecto, los objetivos que nos hemos marcado y también la planificación del trabajo a realizar hasta su conclusión.
- **Capítulo II: Internet de las Cosas**. Este capítulo está dedicado al tema central del proyecto y por tanto veremos todas las bases teóricas necesarias, así como los antecedentes históricos y los principales ámbitos de aplicación del Internet de las Cosas.
- **Capítulo III: Plataformas IoT**. En el siguiente capítulo de este trabajo vamos a exponer todo lo relativo a unas novedosas herramientas que se están desarrollando rápidamente junto con el concepto de IoT, y que merecen un capítulo aparte para poder estudiarlas con más detalle debido a su importancia dentro del desarrollo del IoT durante los últimos años.
- **Capítulo IV: Análisis del producto IoT**. En este capítulo mostraremos en detalle cómo será el producto IoT que vamos a desarrollar como parte práctica de este trabajo, haciendo especial mención a la tarea que queremos solucionar con él y los requisitos que deberá satisfacer.

- **Capítulo V: Diseño del dispositivo IoT**. Este capítulo se centra fundamentalmente en explicar todo lo relativo al hardware que necesitaremos para conseguir nuestro producto IoT, a nivel de electrónica y componentes, y además veremos el proceso que hemos seguido para diseñar y e imprimir en 3D una carcasa física que se añadirá al prototipo final.

- **Capítulo VI: Implementación del dispositivo IoT**. En este capítulo veremos detalladamente cómo se ha implementado la lógica necesaria en nuestra plataforma de desarrollo hardware, además del entorno de desarrollo que necesitaremos instalar y configurar correctamente para conseguir este propósito. Finalizaremos solucionando la comunicación con las dos alternativas de herramienta de gestión y visualización que vamos a emplear en este proyecto y que veremos en el capítulo siguiente.

- **Capítulo VII: Implementación de la Aplicación de gestión y visualización**. Como hemos comentado, en este capítulo veremos cómo utilizar dos alternativas totalmente distintas para solucionar las distintas tareas de gestión que nuestro proyecto requiera y para poder visualizar correctamente los datos recopilados de manera que sean de utilidad para el cliente de la aplicación.

- **Capítulo VIII: Conclusiones.** Para finalizar haremos un breve recopilatorio de los aspectos más importantes que hemos estudiado y realizado a lo largo del trabajo, y finalmente expondremos las posibles mejoras que se puedan llevar a cabo para perfeccionar el producto IoT desarrollado.

- **Anexo A: Lenguaje Unificado de Modelado (*UML*).** Este anexo está dedicado a describir los distintos elementos que forman un diagrama de clases, de forma que el lector pueda entender mejor la estructura del diagrama de nuestro proyecto (ver figura 55).

- **Anexo B: Instalación y configuración de *Android Studio*.** En este anexo se describe el proceso de instalación tanto del entorno de desarrollo como del JDK necesario para desarrollar nuestra aplicación para Android, asi como su configuración inicial.

- **Anexo C: Instalación y configuración de *Arduino IDE*.** De la misma forma que el anexo anterior, en este caso explicamos paso a paso el proceso de instalación del entorno de desarrollo de Arduino y su configuración para poder utilizar en él nuestro kit de desarrollo NodeMCU.

Capítulo II

Internet de las cosas

En este capítulo vamos a explicar qué se entiende por Internet de las Cosas, también conocido por sus siglas *IoT*, del inglés “Internet of Things”. Vamos a desvelar todo lo que conlleva implícito esté término, sobre el cual gira todo este trabajo, para todo aquel que no lo conozca o desee conocer más sobre el concepto del IoT.

Sobra decir que este es un **capítulo fundamental** en este trabajo, pues tiene como objetivo primordial plantar las bases teóricas para la comprensión del trasfondo que nuestro prototipo IoT pretende conseguir.

Comenzaremos con un breve recorrido de la historia de **Internet**, después esbozaremos qué es y cómo funciona la red de redes y destacaremos su **importancia** en prácticamente todos los ámbitos de nuestra sociedad moderna. Para terminar explicaremos el propio **concepto de Internet de las cosas**, desarrollaremos cada una de las **tecnologías habilitadoras** que inciden sobre este concepto a día de hoy y mencionaremos algunos de los **ámbitos de aplicación** más destacados de este paradigma.

2.1. Introducción

Internet es un fenómeno que sin lugar a dudas ha revolucionado el mundo tal y como lo conocíamos hace 30 años. Actualmente es casi indiscutiblemente el **sistema de ingeniería más grande creado por la mano del hombre**, con cientos de millones de computadoras conectadas, enlaces de comunicaciones y switches, y cientos de millones de usuarios que se conectan de forma intermitente a través de dispositivos tan variados como dispositivos móviles, computadoras, tablets, etc. Pero antes de hablar más sobre lo que ha supuesto internet para nuestras vidas vamos a conocer cómo surgió esta tecnología y cómo ha evolucionado hasta nuestros días.

2.1.1. Breve historia de Internet

La primera chispa de la historia de Internet se produce cuando en 1965 cuando el investigador del *MIT*[1] Lawrence G. Roberts consigue conectar por primera vez dos ordenadores a distancia usando una línea telefónica conmutada de baja velocidad, creando así la **primera red de ordenadores de área amplia**, confirmando así las ideas de investigadores anteriores como L. Kleinrock, D. Davies o P. Baran creadores del concepto de *conmutación de paquetes*[2].

En la fase más temprana del desarrollo de Internet se organiza en EEUU la Agencia de Investigación de Proyectos Avanzados, conocida como **ARPA**, perteneciente al Departamento de Defensa de EEUU y con el objetivo del desarrollo de tecnología de vanguardia en el ámbito militar para la defensa nacional.

Entre ellas, una de las más importantes fue la de crear una red de computadoras interconectadas para ser usada como mecanismo de comunicación entre las diferentes instituciones académicas y estatales, con el objetivo de poder tener acceso a la información desde cualquier lugar del país en caso de un ataque enemigo, para lo cual Lawrence G. Roberts se incorpora a ARPA ayudando en

[1] Instituto Tecnológico de Massachusetts es una universidad privada de Cambridge, Massachusetts (EEUU).

[2] Método para agrupar los datos transmitidos a través de una red informática de comunicaciones en paquetes que se componen de un encabezado y una carga útil.

el desarrollo de **ARPANET**, el embrión de lo que hoy conocemos como Internet. Por aquel entonces la red solo se utilizaba como medio de comunicación para fines de investigación entre las Universidades conectadas.

El siguiente hito se produjo debido a las necesidades de mejorar el primer protocolo de comunicación de *ARPANET* (*NCP*[3]), el cual resultaba obsoleto debido al rápido crecimiento que la red estaba experimentando. Por tanto, en 1972 Robert E. Kahn junto a Vint Cerf cooperaron para diseñar y desarrollar una arquitectura abierta de interconexión y desarrollar la nueva generación de protocolos de *ARPANET*, el **protocolo TCP/IP**[4], que se convertiría en el estándar de comunicaciones dentro de las redes informáticas y que, de hecho, se sigue utilizando actualmente.

Para el año 1985 las funciones militares se desligaron de *ARPANET* y esta fue absorbida por **NSFNET** la nueva red creada por la *National Science Fundation* que seguía ligada exclusivamente a propósitos científicos y académicos. A partir de este punto el desarrollo de la red fue abismal, pasando de tener 2.000 ordenadores conectados en el año 1985 a más de dos millones en 1993. (Hart, 2003)

El que quizás sea el hito más importante en la historia de Internet llegó con el desarrollo de la **World Wide Web**, o *WWW*, en 1990 por el científico inglés Tim Berners-Lee y con la ayuda del ingeniero belga Robert Cailliau dentro del *Consejo Europeo para la Investigación Nuclear* (*CERN*).

La *World Wide Web* es un sistema de distribución de documentos de hipertexto o hipermedios interconectados y accesibles por los dispositivos conectados a la red. Hace posible que los usuarios utilicen un navegador web como traductor para visualizar los distintos sitios web que pueden contener texto, imágenes, video, audio y otros contenidos multimedia accediendo a ellos mediante hiperenlaces. El contenido de los mensajes se programa en un lenguaje de

[3] Network Control Program fue el primer protocolo de comunicación empleado en la primera etapa de ARPANET.
[4] Protocolo de comunicación de red en el que se basa Internet y que permite la transmisión de datos entre dispositivos.

hipertexto (*HTML*) que es traducido por el navegador web para mostrar al usuario la página web.

Esta fue la primera aplicación de Internet que atrajo la atención del público general y cambió de forma drástica la forma en que las personas interactúan para comunicarse y buscar información.

Figura 2: Lawrence G. Roberts, R. Kahn, V. Cerf y Tim Berners-Lee reciben el Premio Príncipe de Asturias de investigación científica y técnica en 2002 (Fundación Princesa de Asturias, 2002)

A partir de entonces comienza una etapa en la que Internet crece más que cualquier otro medio de comunicación creándose alrededor de este concepto diversas herramientas y tecnologías que han contribuido al propio crecimiento de Internet, facilitando su utilización para cualquier usuario y añadiendo aplicaciones más innovadoras y útiles que han pasado a ser prácticamente imprescindibles en nuestra vida cotidiana.

Se empieza a utilizar la web para comprar y vender productos o servicios (nace el *e-commerce*) y destaca el gran florecimiento de empresas que nacen o crecen enormemente gracias a las posibilidades que ofrecía Internet que ayudaba a estas empresas a llegar a un gran número de clientes potenciales. Hablamos de grandes empresas como Amazon, eBay, o Google.

Por último, podemos hablar de la fase en que nos encontramos ahora, en la cual predomina la Web “Social” o de las experiencias, gracias a la cual empresas como Facebook o Twitter son muy populares y rentables ya que permiten a las personas comunicarse, conectarse y compartir información sobre ellos mismos

con amigos y familiares (Dave Evans, 2011). Esta idea la desarrollaremos más profundamente más adelante en este capítulo.

2.1.2. ¿Qué es Internet?

Hoy en día Internet se ha convertido en una **herramienta esencial** para nuestras vidas en todos los ambitos. Negocios, relaciones sociales, salud, trabajo, educación, cultura, ciencia, politica, etc. (M. Damas, 2017) Es dificil pensar en algún aspecto de nuestra vida en el que no influya, directa o indirectamente, esta herramienta. Vamos a proceder a conocer un poco mejor los conceptos básicos de esta tecnología para comprender mejor la base sobre la que vamos a trabajar en este proyecto.

Tenemos dos formas de analizar qué es la red de redes. La primera de ellas es describiendo los “engranajes” que forman esta red, es decir, los componentes hardware y software básicos que forman la red. Sabemos que Internet es una **enorme red de computadoras conectadas** entre sí que interconecta cientos de millones de dispositivos informaticos alrededor de todo el mundo. Estos dispositivos, que no hace mucho tiempo se componian principalmente de computadoras PC de escritorio, estaciones de trabajo Linux y los servidores que almacenaban y transmitian información (paginas web, correos electronicos…), son los llamados sistemas terminales o hosts y forman lo que se suele denominar como la **frontera de Internet**. A día de hoy dispositivos como computadores portatiles, dispositivos PDA y smartphones que cuenten con conexión a Internet también forman parte de estos sistemas terminales.

El **núcleo de la red** puede definirse como una extensa malla de conmutadores de paquetes y enlaces por los cuales circula la información (los bits) que transmiten entre sí los sistemas terminales de Internet. El tráfico de los datos por el núcleo de la red es posible gracias a la denominada conmutación de paquetes: el origen divide los mensajes largos en fragmentos de datos mas pequeños que se conocen como paquetes. Estos paquetes viajan entre los enlaces de comunicaciones y los conmutadores de paquetes hasta su respectivo destino, donde los paquetes son reordenados y forman en el sistema terminal destino el mensaje original que partió del origen.

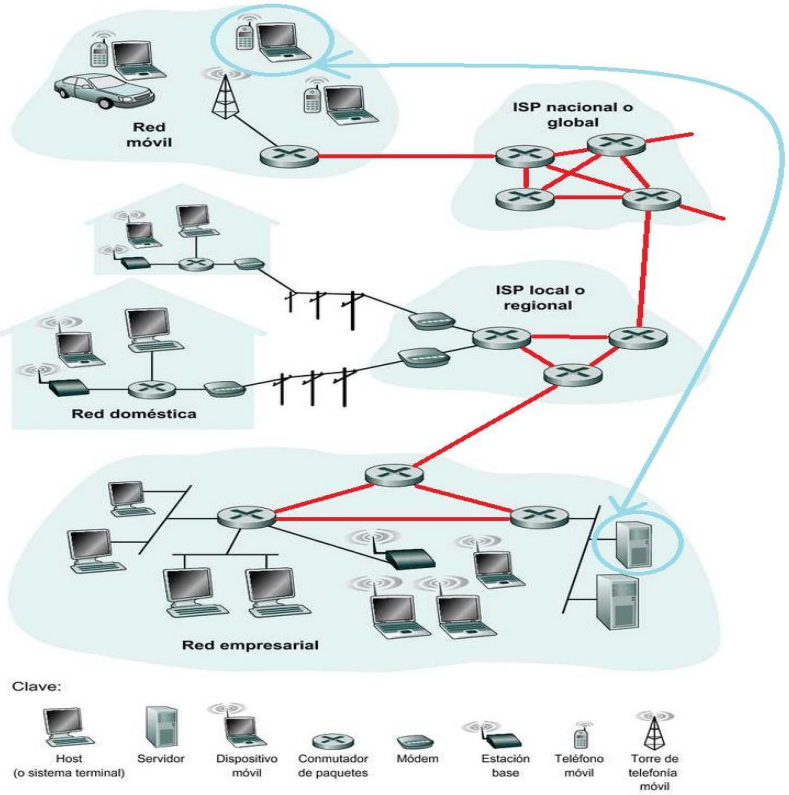


Figura 3: Principales componentes de la red. Sistemas terminales, la interacción que puede existir entre ellos (flecha azul) y el núcleo de la red (resaltado en líneas rojas) (James F. Kurose, 2010)

Otro punto de vista totalmente distinto para analizar Internet es describirlo como la **infraestructura de red que proporciona servicios a aplicaciones distribuidas**. Entre las aplicaciones que más destacan se encuentran el correo electrónico, la navegación web, la mensajería instantanea, voz sobre *IP* (*VoIP*), radio a través de Internet, flujos de video, juegos distribuidos, compartición de archivos en redes entre iguales o entre pares (*P2P*, peer-to-peer), televisión a través de Internet (*IPTV*) y las sesiones remotas.

Al desarrollar una aplicación distribuida, tenemos que tener en cuenta que los programas que se ejecuten en los distintos sistemas terminales tendrán que enviarse datos entre sí. Para ello, los sistemas terminales conectados a Internet proporcinan una **API** (Application Programming Interface) que especifica como un programa de software que se ejecuta en un sistema terminal pide a la infraestructura de Internet que suministre datos a un programa de software de destino específico que se ejecuta en otro sistema terminal. Una *API* no es más que un **conjunto de reglas que nuestra aplicación debe cumplir para**

transmitir los datos a través de Internet al programa de destino que se ejecuta en otro sistema terminal distinto al de origen.

Como analogía podemos pensar que se trata de un proceso similar al de enviar una carta por correo ordinario. Para llevar a cabo esta tarea no basta con escribir la carta (los datos) y lanzarla al aire. Hay que seguir una serie de pautas que la empresa de mensajería dispone para que la carta pueda ser enviada y entregada a su destinatario correctamente (introducir la carta en un sobre, escribir el nombre completo del destinatario, su dirección y código postal, después cerrar el sobre y pegar un sello en la esquina superior derecha y, por ultimo, introducir el sobre en un buzón del servicio de mensajería). Así, el servicio de mensajeria tiene su propia "*API* de servicio postal" que cualquiera que desee emplear el servicio deberá seguir, al igual que Internet tiene una *API* que el programa que envía los datos debe seguir para que Internet entregue el mensaje al sistema y software de destino.

Esta segunda definición de Internet es muy importante, pues Internet es una infraestructura en la que constantemente se estan creando e implementando nuevas aplicaciones y, cada vez más, las necesidades de estas aplicaciones están dirigiendo los avances de los componentes esenciles de Internet de los que hemos hablado en nuestra primera definición.

2.1.3. La importancia de Internet en nuestros días

A lo largo de sus carreras, los denominados "padres de Internet" han recibido multitud de honores y premios por haber diseñado y realizado un sistema que ha cambiado el mundo al ofrecer **posibilidades antes impensables para el progreso científico y social**. Hoy en día para nosotros ya no se trata de un fenómeno nuevo, pero es indudable que Internet ha revolucionado el mundo tal y como se conocía hace apenas tres décadas. Internet se ha vinculado estrechamente con casi cualquier ambito de nuestra sociedad, sobre todos con aquellos ligados estrechamente con la comunicación (C. Perera, 2014).

A dia de hoy es una tecnología **imprescindible** que ha proporcionado un medio donde no existen barreras para la comunicación, sin tener en cuenta el espacio,

fronteras, distancias o sociedades. En **cualquier ámbito se puede comprobar el impacto profundo que ha supuesto Internet**: trabajo, entretenimiento, educación, compras, salud, finanzas, negocios, etc. En definitiva ha conseguido hacer nuestra vida un poco más facil en muchos aspectos llegando a un punto en el que hoy en día no concevimos una sociedad moderna sin las herramientas y las posibilidades que nos brinda Internet.

2.1.4. El despertar del Internet de las Cosas

Como hemos visto anteriormente en este capítulo, el desarrollo de Internet, y de la tecnología en general, ha sido muy veloz e intenso, llegando a estar presente a nivel mundial en poco menos de tres décadas de historia. En sus inicios, Internet comenzó siendo una red de computadoras, pero **conforme los avances tecnológicos lo han permitido se han ido añadiendo nuevos dispositivos y nuevas formas de interactuar** con la red que forman lo que a día de hoy es Internet.

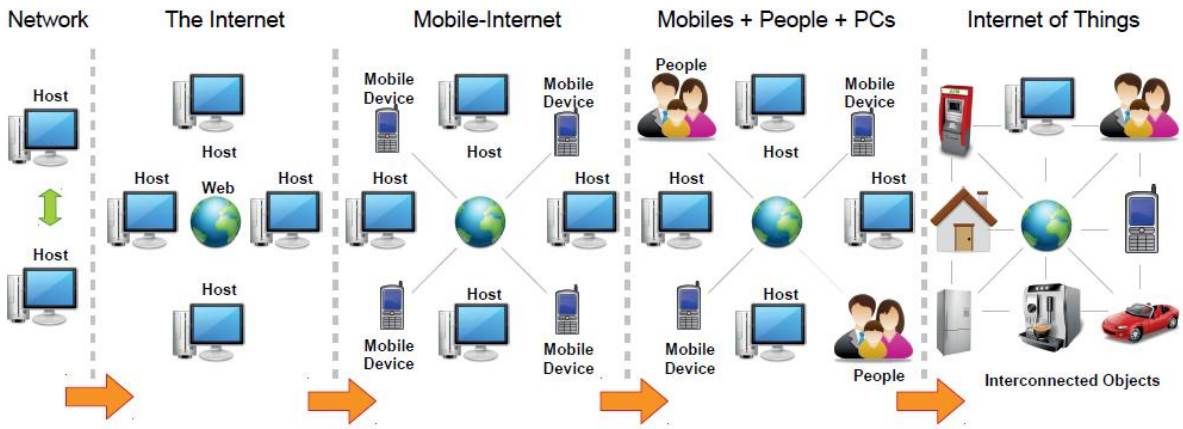


Figura 4: Los actores que se han ido añadiendo a Internet desde sus inicios

El acceso a Internet desde los **dispositivos móviles** fue una de las primeras nuevas formas que los usuarios tenían para navegar por la red y utilizar todo su potencial, fuera de las típicas computadoras de escritorio tal y como tradicionalmente se conocían, en incluirse en lo que sería el repertorio de métodos para acceder a la red. Esto ayudó a romper aún más cadenas en beneficio de los usuarios, ya que permitía a cualquiera disponer de acceso a la red en cualquier lugar donde dispusiera de un smartphone y cobertura suficiente,

eliminando la necesidad de tener que estar anclados a las computadoras tradicionales, que pecan de tener una casi inexistente movilidad. Muy importante fue para este caso el auge de las aplicaciones de mensajería instantanea que facilitó enormemente la comunicación entre conocidos que antes de esto tenian que acudir a las llamadas telefónicas o a los más costosos mensajes de texto. Esto hizo que las ventas de este tipo de dispositivos crecieran de forma exponencial durante los años venideros.

Los siguientes nuevos actores que se presentaron en Internet fueron **las propias personas a través de las redes sociales**. Las redes sociales son sitios o aplicaciones web formados por comunidades de individuos con intereses o actividades en común, que permiten el contacto entre estos, de manera que sirven como punto de encuentro donde se puedan comunicar, interactuar e intercambiar información entre sus miembros de un manera rápida, sencilla y sin coste alguno. En el año 2003 aparecieron algunos de los sitios más populares que lograron hacer crecer exponencialmente el uso de este tipo de servicios, tales como *MySpace* o *Friendster*, entre otras.

Sin embargo el gran boom de estos sitios llegó de la mano de la red social por antonomasia, *Facebook*. Desde septiembre de 2006, cuando *Facebook* se abre a todos los usuarios de Internet (anteriormente era un sitio destinado exclusivamente a estudiantes) este tipo de sitios web se popularizaron entre el gran público. Este éxito dio lugar a la creación de distintos tipos de redes sociales diferenciadas entre ellas pero que comparten la misma base, como *Twitter* o *Instagram*, que también son muy populares a día de hoy, sobre todo entre el público más joven. Existen redes sociales clasificadas según su uso, para profesionales (*LinkedIn*), ciencia y educación (*Academia.edu*), y aficionados de diversos ambitos como la fotografía (*Pinterest*, *Instagram*,…), la música (*Soundcloud*), videos (*YouTube*, *Vimeo*,…), viajeros (*TripAdvisor*), etc. Los distintos tipos de redes sociales suponen uno de los principales atractivos de la web para los usuarios: *Facebook* cuenta con 2.000 millones de usuarios activos al mes, *Twitter* posee 317 millones e *Instagram* alrededor de los 600 millones, sólo por mencionar algunas de las redes sociales más comunes.

En estos ultimos años han sido los mismos **objetos cotidianos** que todos utilizamos en nuestra rutina diaria los que se han ido incorporando a Internet. Nos referimos a objetos tales como grandes y pequeños electrodomesticos, lamparas, accesorios de vestir, ropa, vehiculos, utensilios de cocina, muebles, y un sinfín de posibilidades. Este fenómeno ha dado lugar a un concepto muy contemporáneo denominado **Internet de las Cosas** o ***IoT***, por sus siglas en inglés “Internet of Things”. El concepto del *IoT* se le atribuye al investigador del *MIT* Kevin Ashton, que describió en 1999 un sistema donde Internet se conecta al mundo físico real a través de una serie de sensores que facilitan información sobre el entorno. Se podría definir, por tanto, el Internet de las Cosas como la **convergencia en la evolución de distintos tipos de tecnologías hardware y software que están permitiendo que cada vez más objetos heterogeneos se puedan interconectar entre sí, dotandolos de una mayor inteligencia, y permitiendo además crear nuevos servicios y oportunidades de negocio** (M. Damas, 2017). Se puede ver como una consecuencia de la evolución que ha sufrido Internet en estos ultimos diez años, en los cuales hemos vivido una nueva forma de uso de Internet donde todo se ha convertido en social, transaccional y movil, dejando atrás el uso que se le daba a Internet hace dos décadas, que se limitaba basicamente a una herramienta para buscar información.

Figura 5: Kevin Ashton, investigador del MIT quien acuñó el concepto de IoT en 1999

Para poner en situación la potencia de este nuevo concepto ligado a Internet podemos comprobar que desde hace cuatro años la consultora americana *Gartner*, que hace estudios anuales en los que representa gráficamente mediante una curva la evolución de las principales tecnologías emergentes, considera el Internet de las Cosas como una de las tecnologías con mayores

perspectivas de crecimiento a nivel mundial, situándolo en el pico de gran expectativa, como se puede apreciar en la siguiente figura.

Figura 6: Ciclo de sobre expectación de tecnologías emergentes (Gartner, 2015)

Este ciclo de sobreexpectación de *Gartner* es una representación gráfica de la madurez, adopción y aplicación comercial de tecnologías específicas. Se utiliza desde 1995 para caracterizar el entusiasmo sobredimensionado y la subsiguiente decepción que ocurre habitualmente en la introducción de nuevas tecnologías (Wikipedia, 2018)

A pesar de esto, en el último ciclo de sobre expectación publicado por *Gartner* en julio de 2017 (ver figura 7) se puede apreciar que ya no aparece explícito el Internet de las Cosas, lo cual puede tener varias interpretaciones. Puede ser que **se haya considerado que otros conceptos** como el hogar conectado o las *Plataformas IoT* **los representan**, ya que están íntimamente relacionados con el *IoT*. Por otra parte puede ser que **ya sea considerada como una tecnología consolidada** y, por tanto, se haya dejado de tener en cuenta como una expectativa sino una realidad establecida. Por último podríamos considerar que, **sencillamente no han sabido cómo ubicarla correctamente dentro de la**

curva del ciclo de sobre expectación, por lo que no aparece explícitamente. Sin embargo, esto no cambia la realidad de que este concepto sigue siendo muy popular a día de hoy, como podemos comprobar también analizando las **tendencias de búsqueda** de *Google* (ver figura 8), donde se observa que el concepto de Internet de las Cosas ha crecido en popularidad desde los últimos cuatro años, junto con otros conceptos también relacionados.

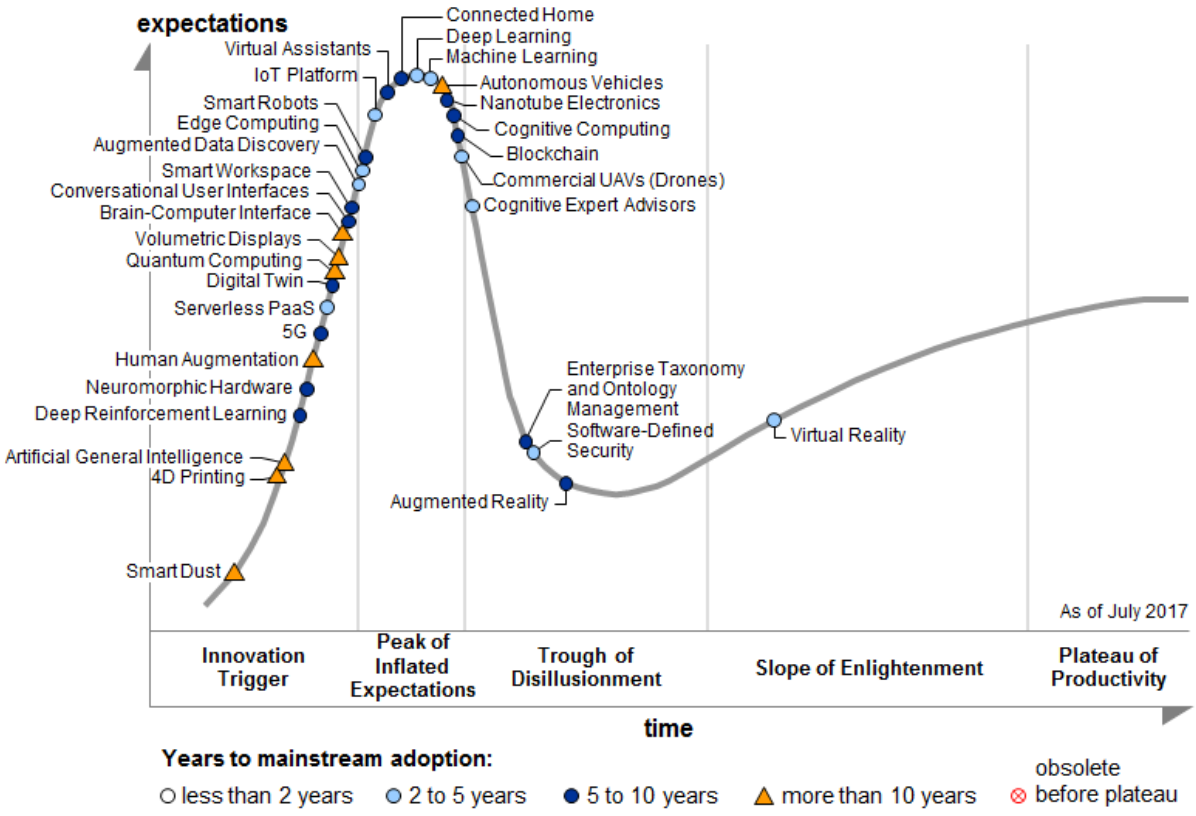


Figura 7: El último ciclo de sobre expectación publicado por Gartner, donde no aparece el IoT pero sí otros conceptos relacionados (Gartner, 2017)

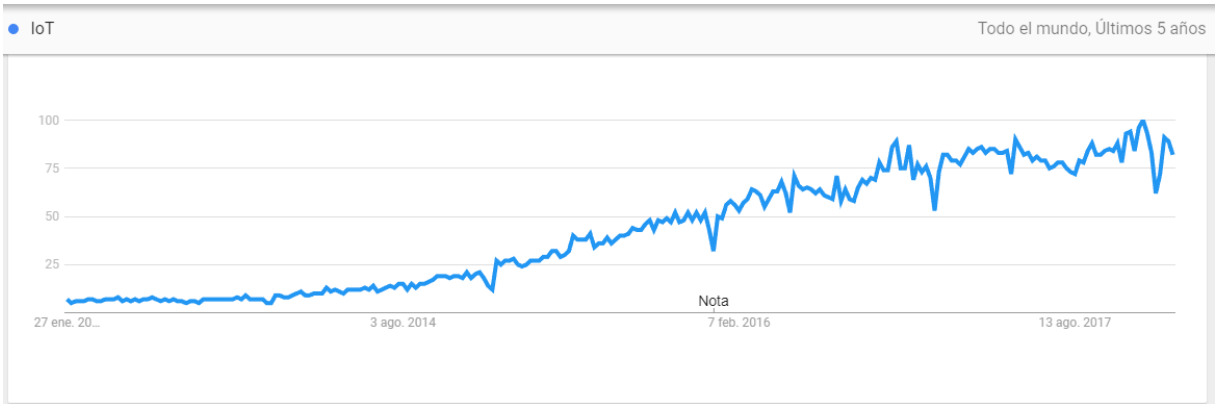


Figura 8: Representación gráfica de las búsquedas en Google del término IoT durante los últimos 5 años (Google, 2018)

Otro indicio de la popularidad del *IoT* es analizarlo desde el **punto de vista económico**, pues los ingresos globales del Internet de las Cosas también estan creciento de manera muy significativa actualmente y todas las predicciones apuntan a que esta escala de progreso se va a mantener e incluso se va a incrementar a lo largo de los proximos meses (Vazhnov, 2015).

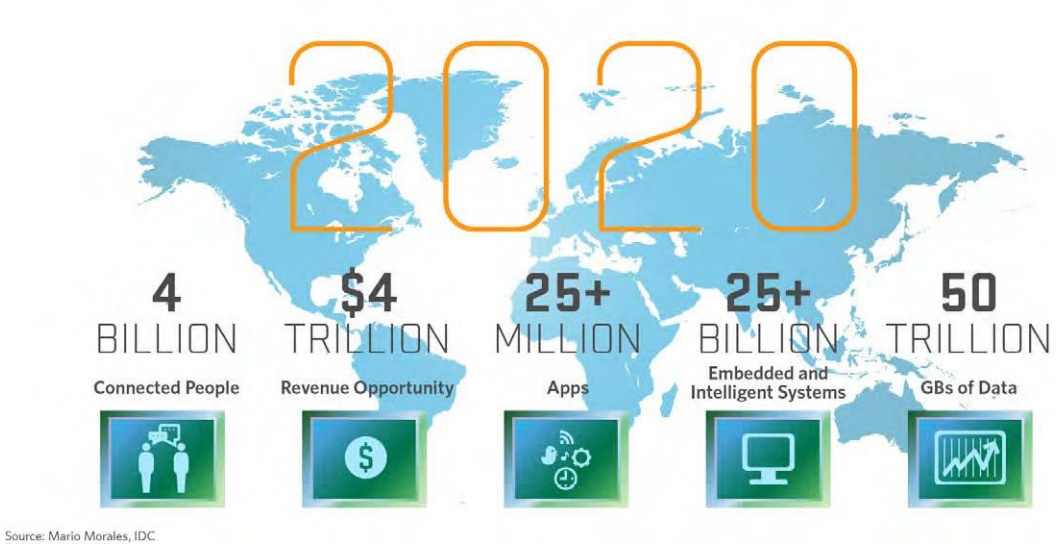


Figura 9: Predicciones sobre el IoT en 2020 (IDC)

Grandes empresas multinacionales como *Cisco* y *Ericsson* están apostando fuertemente por este concepto y predicen que para el año 2020 habrá más de 25.000 millones de objetos (“things”) conectados en el mundo, con el consiguiente impacto que este hecho supondría tanto a nivel social como a nivel económico. Solamente este año el volumen de negocio de las cosas conectadas a internet ascenderá a 475.000 millones de euros. Con estas previsiones de crecimiento el Internet de las Cosas alcanzaría un negocio mundial de 772.5 billones de dólares en 2018, un 14.6% más que el año pasado, llegando a superar el trillón de dólares para el año 2020 por la mayor productividad, ahorro de costes y nuevos mercados para las empresas. (IDC, Diciembre 2017).

Por poner un ejemplo de algo que ya es una realidad dentro del marco económico relacionado con el Internet de las Cosas, podemos remarcar que sólo en el año 2016 se produjeron una serie de adquisiciones muy significativas relacionadas con el *IoT*:

- ***Qualcomm*** compró *NXP* por 47.000 millones de dólares.

- ***SofBank*** adquiere *ARM* por 31.000 millones de dólares.
- ***Cisco*** compra *Jasper* (una *Plataforma IoT*) por 1.400 millones de dolares.
- ***TDK*** adquiere *InvenSense Inc*. (fabricante de sensores para el *IoT*) por 1.300 millones de dolares.
- ***Cypress Semiconductor*** compra *Broadcom* por 550 millones de dólares

Es inegable, por tanto, el impacto económico que tendrá (y ya está teniendo) el Internet de las Cosas, creando **nuevos modelos de negocio** que se beneficiarán de las infinitas posibilidades que nos propone esta tecnología.

En cuanto al **impacto en investigación**, se puede observar también que actualmente se esta produciendo un incremento exponencial de publicaciones donde aparece el término "Internet of Things" dentro de las bases de datos donde se encuentran las publicaciones científicas con prestigio a nivel internacional (ver figura 10). Es destacable ademas señalar que en los documentos que se acaban de publicar del "Horizonte 2020", donde se marcan las líneas de investigación que se van a incentivar a nivel europeo, se puede comprobar que el Internet de las Cosas tiene una **posición muy destacada**. También **en nuestro país** ya se esta viendo la importancia que esta cobrando el *IoT*, con hechos como los que acontecieron el año pasado cuando el gobierno, y más concretamente el exministro de Fomento Íñigo de la Serna, aprobó un plan para potenciar la digitalización, la transformación energética y el Internet de las Cosas.

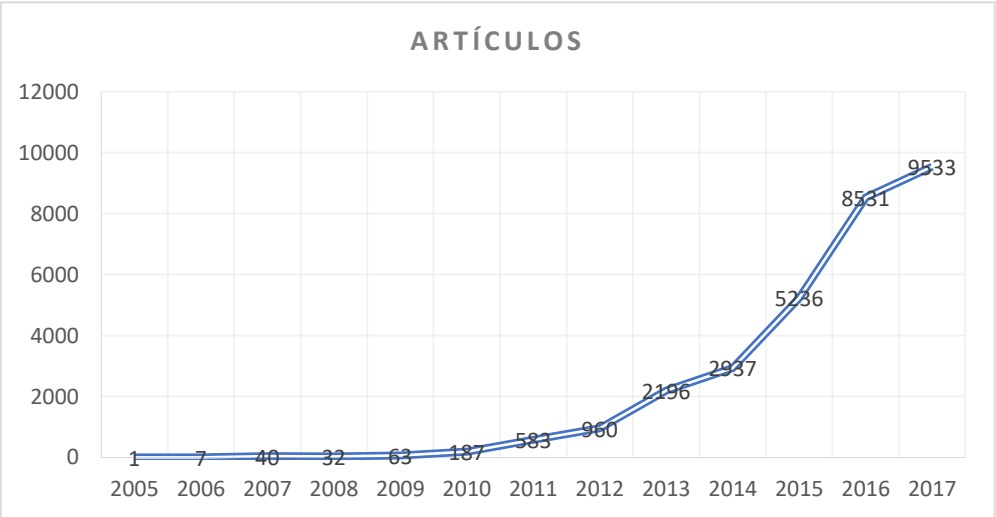


Figura 10: Impacto del IoT en investigación

En conclusión, podriamos decir que muchos investigadores y técnicos pertenecientes a grandes empresas consideran el Internet de las Cosas como **la proxima evolución de Internet** que lo está cambiando todo, incluido a nosotros mismos (A. McEwen, 2014).

Esto puede parecer una afirmación algo atrevida, pero si recordamos la etapa de *ARPANET* que hemos comentado brevemente al principio de este capítulo y lo comparamos con lo que es Internet a día de hoy podemos sacar como conclusión que **esta tecnología ha sufrido un proceso continuo de desarrollo y mejora**, pero en esencia hace la misma función que en aquellos inicios: permitir la comunicación y el intercambio de información entre sus usuarios. Por esta razón puede verse el impacto del Internet de las Cosas como una **verdadera evolución** de Internet, pues va a permitir aumentar enormemente sus capacidades relacionadas con la recopilación, el análisis, la distribución y, sobre todo, el poder convertir estos datos en información, conocimiento y, lo más importante, sabiduría.

En este contexto podemos hablar de que el *IoT* cobra una gran importancia porque es una evolución real, **un salto de calidad que abrirá nuevas posibilidades de desarrollo para aplicaciones revolucionarias** que permitirán mejorar significativamente la manera de vivir, aprender, trabajar y entretenerse de todos sus usuarios. Sólo con los avances que se han conseguido a día de hoy ha logrado que Internet sea sensorial (temperatura, humedad, presión, vibración, etc), lo que nos permite ser más proactivos en lugar de reactivos.

Hoy en día el Internet de las Cosas está en **pleno proceso de desarrollo**, y a la vez el número de dispositivos conectados crece cada día que transcurre. Como ya hemos mencionado anteriormente, se estima que para el 2020 existirán más de 25.000 millones de objetos conectados a Internet, y es importante aclarar que estas estimaciones no tienen en cuenta los rápidos avances que se producen en la tecnología de Internet o de los dispositivos, y que las predicciones se basan obviamente en los conocimientos que tenemos hoy en día. Por tanto, gracias a las nuevas capacidades de la próxima evolución de Internet que supone el Internet de las Cosas podremos detectar, recopilar, transmitir y analizar datos a

escala masiva, lo cual combinado con la facilidad de procesar información de las personas, permitirá que la humanidad tenga el conocimiento y la sabiduría necesaria para prosperar en los proximos meses, años, décadas y siglos.

Figura 11: Los seres humanos convierten los datos en sabiduría (Evans, 2011)

Cabe mencionar que esta evolución ya la predijo Mark Weiser, uno de los principales ideólogos de la computación ubicua, en el año 1991 cuando afirmaba que en pocos años tendríamos tantas cosas conectadas a nuestro alrededor que no nos daremos ni cuenta que Internet estará ahí.

Vamos a proceder ahora a comentar algunas circunstancias que han contribuido al desarrollo del Internet de las Cosas, que son tan importantes, o incluso más, que las propias tecnologías habilitadoras que describiremos más adelante.

- En primer lugar, es clave el **abaratamiento de la tecnología** para conseguir que el IoT sea una realidad. En los ultimos años se ha conseguido que el coste necesario para poder poner un dato en circulación a través de Internet sea lo suficientemente pequeño como para que no suponga una barrera a la hora de desarrollar un producto IoT.
- En segundo lugar, la posibilidad de usar **fuentes de energía alternativas para los dispositivos conectados a Internet**, gracias a los avances conseguidos en los ultimos años relacionados con el concepto de *Energy Harvesting* (Sandhya Chalasani, 2008) que tienen como objetivo conseguir que dichos productos *IoT* puedan autoabastecerse energéticamente.

- Tambien hay que mencionar los **nuevos modelos de negocio emergentes** impulsados por las estrategias de mecenazgo y financiación colectiva como las aceleradoras y el *crowfounding*, que fomentan la creación de startups y empresas.
- Y por último también **ciertas políticas gubernamentales** en algunos países y ciudades que han financiado proyectos relacionados sobre todo con las *Smart Cities* para mejorar temas como el tráfico, la iluminación, la polución, etc.

A continuación vamos a describir más detalladamente algunas de las **tecnologías habilitadoras o potenciadoras más importantes** (O. Vermesan, 2013) que están siendo claves para este fenómeno que, como ya hemos visto, está tan de moda y revolucionando el mercado económico.

2.2. Placas de desarrollo

Una de las partes más importantes a la hora de desarrollar un producto IoT es la elección de una placa de desarrollo, es decir, el **dispositivo hardware electrónico donde se implementan los diferentes sensores, actuadores y la unidad de procesamiento que se encargará de la lógica de nuestra aplicación**. Es lo que se suele denominar dispositivo conectado o dispositivo inteligente, es decir, un dispositivo electrónico conectado, por lo general, a otros dispositivos o a redes a través de diversos protocolos como pueden ser Bluetooth, NFC, Wi-Fi, 3G, X10, etc, cuya característica principal es que puede funcionar hasta cierto punto de forma interactiva y autónoma.

Placas o plataformas de desarrollo existen muchas, pero aquí vamos a destacar algunas de las mas conocidas a día de hoy.

2.2.1. Arduino

Arduino es un **proyecto open source que ha adquirido una gran popularidad** en los ultimos años. Cuenta con una **amplia comunidad internacional** y la propia empresa se encarga de diseñar y manufacturar sus propias placas de

desarrollo hardware para construir dispositivos inteligentes que puedan monitorizar y controlar objetos del mundo real de una manera sencilla y accesible. Las placas *Arduino* estan disponibles comercialmente en forma de placas ensambladas o también en forma de kits DIY (*Do It Yourself*).

Existe una gran diversidad de placas hardware *Arduino* diferentes que emplean una gran variedad de microcontroladores y microprocesadores, aunque generalmente suelen incluir un microcontrolador *Atmel AVR* conectado sobre una placa de circuito impreso, sobre la que es posible añadir distintas **placas de expansión** (*Shields*) a traves de los diferentes puertos de entrada y salida presentes en la placa base. Estos *Shields* **complementan la funcionalidad del modelo concreto de Arduino** agregando facilmente circuiteria, sensores, y modulos de transmisión externos a la placa principal (por ejemplo podemos añadir un shield *Ethernet* o *WiFi* para permitir la comunicación de la placa base a través de estos protocolos de comunicación). Suelen alimentarse por un puerto *USB* y pueden ser programadas a traves del puerto serie que incorporan haciendo uso del *Bootloader* que incluyen por defecto.

Para programar un *Arduino* se suele emplear el lenguaje *C++*, aunque es posible programarlo en otros lenguajes como *Java*, *Python*, *PHP*, *Ruby*, etc. *Arduino* además provee una serie de librerias que facilitan la programación del microcontrolador, que se pueden consultar en su sitio web oficial. Una de las claves del éxito de *Arduino* se debe a que nos permite programar un **MCU**[5] abstrayendonos de las complejidades que acompañan al microcontrolador específico, gracias a unas herramientas sencillas y específicas para programar aquellos microcontroladores que emplean sus placas.

Para facilitar aun más el trabajo con estas placas, *Arduino* nos ofrece un **entorno de desarrollo integrado** (*IDE*) con el que podemos programar sus placas de una forma muy sencilla. Este *IDE* (que se describirá con detalle en el capítulo VI) además de permitir la programación de la placa que conectemos mediante el puerto *USB*, nos permite descargar y añadir las librerias y componentes establecidos por *Arduino* de forma oficial, e incluso incorpora un **gestor de**

[5] MicroController Unit es un circuito integrado programable, capaz de ejecutar las órdenes grabadas en su memoria.

placas que nos permite descargar *frameworks* para la programación de **otras placas de desarrollo distintas a las propias de *Arduino***.

Como ya hemos comentado, existen distintos modelos de placas *Arduino* que difieren entre sí en componentes y características como el microcontrolador, voltaje de funcionamiento, pines *I/O* digitales, pines de entradas analogicas, memoria *flash*, etc. Entre las placas *Arduino* más populares se encuentran:

- *Arduino Yún*
- *Arduino Due*
- *Arduino Mega ADK*
- *Arduino Ethernet*
- *Arduino Robot*
- *Arduino Leonardo*
- *Arduino Zero*

Las placas *Arduino* se comercializan con precios que empiezan desde los 15€ aproximadamente.

Figura 12: Diferentes versiones comerciales de placas Arduino

2.2.2. Raspberry PI

Raspberry Pi es un **computador de placa reducida**, computador de placa única o computador de placa simple (*SBC*) de bajo costo desarrollado en Reino Unido por la *Fundación Raspberry Pi*, con el objetivo de estimular la enseñanza de ciencias de la computación en las escuelas. Es un producto con propiedad registrada, pero **se permite su uso libre tanto a nivel educativo como particular**. Su software sí es open source, dispone de un sistema operativo oficial basado en *Debian*, denominado *Raspbian* y además permite también utilizar distintos sistemas operativos, incluso una versión de *Windows* 10 para *IoT*.

En todos sus modelos incluye un procesador *Broadcom*, una memoria *RAM*, una *GPU*, puertos *USB*, *HDMI*, *Ethernet* (excepto el primer modelo), 40 pines *GPIO*[6] y un conector *RCA*[7] (sólo los primeros modelos). Ninguna versión incorpora memoria, siendo la unica opción en su primera versión emplear una tarjeta *SD* y en ediciones posteriores una tarjeta *MicroSD*.

La *Fundación Raspberry Pi* fomenta principalmente el uso del lenguaje de programación *Python*, siendo también soportados otros lenguajes como *Tiny BASIC*, *C*, *Perl* y *Ruby*.

Los diferentes modelos de Rasperry Pi que se han lanzado al mercado son:

- *Raspberry Pi 1 Modelo A*
- *Raspberry Pi 1 Modelo B y B+*
- *Raspberry Pi 2 Modelo B*
- *Raspberry Pi 3 Modelo B*
- *Raspberry Pi 3 Modelo B+*

[6] Pin de Entrada/Salida de propósito general
[7] Conectores para video compuesto y sonido estereofónico, actualmente en desuso.

Figura 13: Raspberry Pi 3 Modelo B+ presentado en marzo de este año

Aparte de los modelos normales, la *Fundación Raspberry Pi* ha lanzado otra gama de placas denominadas *Raspberry Pi Zero*, mucho **más pequeñas y algo menos potentes que las anteriores**, especialmente indicadas para el desarrollo de productos *IoT*. Concretamente son los modelos *Rasperry Pi Zero* y *Pi Zero W* (Wireless) que incluye *WiFi* y *Bluetooth*.

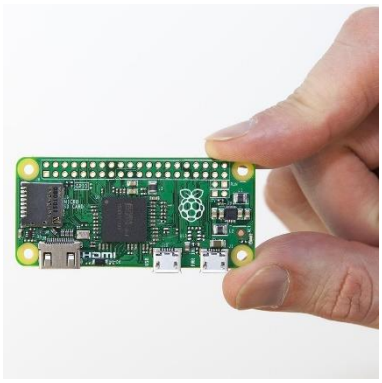


Figura 14: La nueva gama Raspberry Pi Zero, tiene un tamaño de 65 x 30 mm

El modelo de referencia actualmente es el nuevo *Rasperry Pi 3 Modelo B+* presentada el 14 de marzo de 2018 cuyas mejoras técnicas introducidas son breves con respecto al modelo anterior, destacando un procesador ligeramente más rápido (unos 200 MHz concretamente), un módem *WiFi* 802.11ac, la actualización de la conectividad *Bluetooth* a la versión 4.2 y un nuevo cabezal de cuatro conectores para Power over Ethernet[8] (*PoE*). Se suele encontrar con un PVP oficial de unos 40€, mientras que el modelo *Pi Zero W* ronda los 25€.

[8] Alimentación a través de Ethernet es una tecnología que incorpora alimentación eléctrica a una infraestructura LAN estándar, permitiendo que se suministre energía a un dispositivo de red mediante el mismo cable que se utiliza para la conexión de red.

2.2.3. ESP8266

El *ESP8266* es un chip que fue lanzado al mercado originalmente en 2014 con el módulo *ESP-01*, pensado para actuar como un adaptador de puerto serie a *WiFi* por un precio muy asequible. Fue desarrollado por la empresa china *Espressif Systems*, con sede en Shangai y consiguió rápidamente una gran popularidad empleandose principalmente como alternativa a los otros shields para *Arduino* de mayor coste.

Sin embargo, gracias a las peticiones de la comunidad de desarrolladores que creían en su potencial como *MCU* independiente, meses después de su lanzamiento la empresa fabricante del *ESP8266* proporcionó una ***API* para desarrollar directamente sobre su microprocesador**, pudiendo emplearse de este modo el chip de manera independiente. No contentos con esto, la comunidad que utilizaba el *ESP8266* para sus proyectos consiguió adaptar su *API* para poder **utilizarse en el entorno de Arduino**, ofreciendo a los desarrolladores que programasen el *ESP8266* la capacidad de utilizar las librerias y el conocimiento acumulado con el uso de *Arduino*.

Gracias a estas aportaciones, hoy en día existen varios *SDK* de código abierto para el *ESP8266*, entre los que encontramos un firmware basado en C++ para Arduino, lo que permite que el *MCU* del *ESP8266* y sus componentes sean programados, como cualquier otro dispositivo *Arduino*, fácilmente a traves de su *IDE*.

Entre sus **características** (Espressif Inc, 2017) se encuentran:

- Un **procesador** RISC[9] Tensilica Xtensa LX106 de 32-bit con una frecuencia de 80 MHz ampliable hasta 160 MHz RISC
- **RAM de instrucción** de 64 KB
- **RAM de datos** de 96 KB
- **Capacidad de memoria externa** flash QSPI de hasta 16 MB
- **WiFi** estandar IEEE 802.11 b/g/n

[9] Tipo de diseño de CPU utilizado en microprocesadores con instrucciones de tamaño fijo presentadas en un reducido número de formatos y en los que sólo las instrucciones de carga y almacenamiento acceden a la memoria de datos*.

- 16 pines **GPIO**
- Alimentación de **3.3V**.

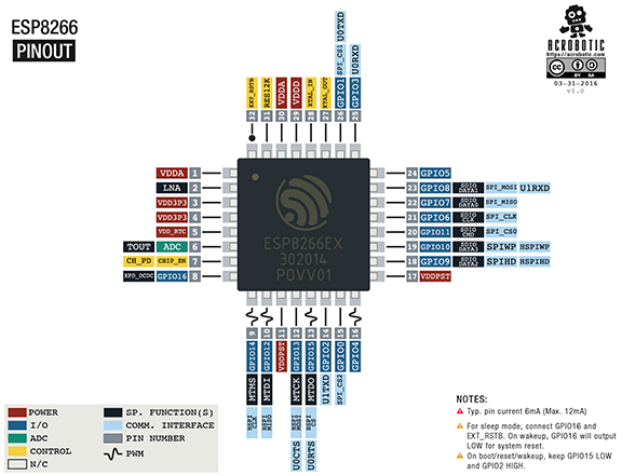


Figura 15: Disposición de los pines del ESP8266

Cabe destacar que el tamaño del ESP8266 es muy reducido, gracias a que utilizan para su fabricación una tecnologia denominada *System on a Chip* o *SoC* en la que se integran todos o gran parte de los modulos que componen un computador o cualquier otro sistema informático o electrónico en un único circuito integrado. Esta característica es **ideal para el uso de esta plataforma de desarrollo en productos IoT**.

Existen diversos módulos *ESP8266* que difieren en características. En principio cualquiera de ellos puede programarse utilizando los lenguajes que hay disponibles para ellos y mediante el *IDE* de *Arduino* como hemos mencionado anteriormente. Por tanto la elección del módulo dependerá principalmente de la implementación hardware que más se adecue a nuestras necesidades, teniendo en cuenta que la parte que más nos podría interesar es la **cantidad de memoria disponible**. Entre los distinos modulos se encuentran el ESP-01, ESP-02, ESP-07, ESP-12F, etc. Se suelen encontrar con un PVP aproximado de 5€.

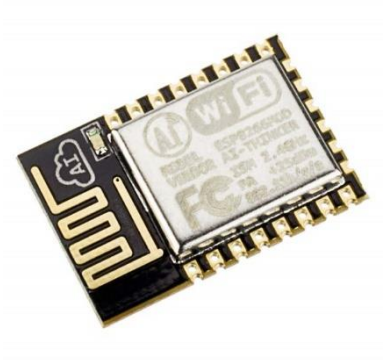


Figura 16: Módulo ESP-12 que integra un ESP8266, es similar en tamaño a una moneda

A pesar de las ventajas que supone el uso del *ESP8266* como placa de desarrollo, especialmente para proyectos relacionados con el Internet de las Cosas, este chip tiene diversas **limitaciones** que ocasionan ciertas restricciones para su uso en desarrollos más generales pues su propósito inicial no era más que servir como un adaptador de serie a *WiFi* de bajo coste. Por esta razón, a pesar de que estas limitaciones suelen ser salvables en la mayoría de los casos, la empresa *Espressif* comenzó en 2015 a trabajar en un nuevo producto con nuevas características a petición de la comunidad de desarrolladores del *ESP8266* y debido principalmente a la popularidad que este chip había alcanzado gracias a su precio y sus posibilidades para el desarrollo de dispositivos conectados. A finales de ese mismo año la empresa anunció el lanzamiento de la versión preliminar del **nuevo chip denominado ESP32** y tras un periodo de testeo se lanzó la versión definitiva al mercado en septiembre de 2016.

El modelo *ESP32* esta diseñado como microcontrolador para aplicaciones del Internet de las Cosas siendo más rápido y potente que su antecesor, vamos a repasar sus **principales características** (Espressif Inc, 2018):

- Integra un **procesador** Xtensa Dual-Core LX6 con arquitectura Harvard x32 y una frecuencia de hasta 240 MHz, lo cual es una característica muy interesante pues al incluir dos núcleos **se puede emplear uno de ellos a las comunicaciones y el otro al resto de procesos**, resolviéndose así una de las limitaciones más importantes del modelo ESP8266.

- Cuenta con un **coprocesador adicional de muy bajo consumo** asociado al *RTC*, dedicado para ciertas tareas (por ejemplo, la comprobación de pines de entrada/salida) mientras el chip se encuentra en estado de hibernación.
- Cuenta con una **memoria SRAM** interna de 520 KB accesible por ambos núcleos y ampliables mediante memoria SRAM externa de hasta 8 MB.
- Cuenta con una **memoria ROM** interna de 448 KB para el Bootloader y funciones internas, una novedad con respecto al modelo anterior.
- **Soporta memoria flash** externa de hasta 16 MB.
- Cuenta con **pines** *GPIO*, *PWM*[10] y *ADC*[11] al igual que el *ESP8266* pero en mayor número.
- Añade **conversores** *DAC*[12] y canales *PWM* dedicados para el control de LEDs y motores.
- Soporta **protocolos** *I2C*, *serie*, *I2S*, *SPI* e *IR*, al igual que su antecesor.
- Añade **nuevas interfaces**, tales como *Ethernet*, *SDIO*, *CAN* bus 2.0 y *Bluetooth* 4.2 ampliando las posibilidades de comunicación enormemente.
- **Sensor de efecto *Hall***[13] y un **termómetro**.
- Incluye un acelerador de **encriptación por hardware** que implementa seis algoritmos del estándar *FIPS PUB 197* (concretamente son *AES-128*, *AES-192* y *AES-256* tanto en modo de encriptación como para desencriptar), lo que supone una mejora significante en cuanto a la seguridad de las comunicaciones con respecto al *ESP8266*.

[10] Modulación por ancho de pulsos (Pulse-Width Modulation) se utiliza para emular una señal analógica (por ejemplo, para regular la intensidad de un iluminador LED).
[11] Analog to Digital Conversion es una entrada que convierte valores analógicos en digitales para su procesamiento por parte del microprocesador.
[12] Digital to Analog Conversion, tiene una función similar a los pines PWM.
[13] Se conoce como la aparición de un campo eléctrico por separación de cargas en el interior de un conductor por el que circula una corriente en presencia de un campo magnético con componente perpendicular al movimiento de las cargas. Este campo eléctrico es perpendicular al movimiento de las cargas y a la componente perpendicular del campo magnético aplicado*.

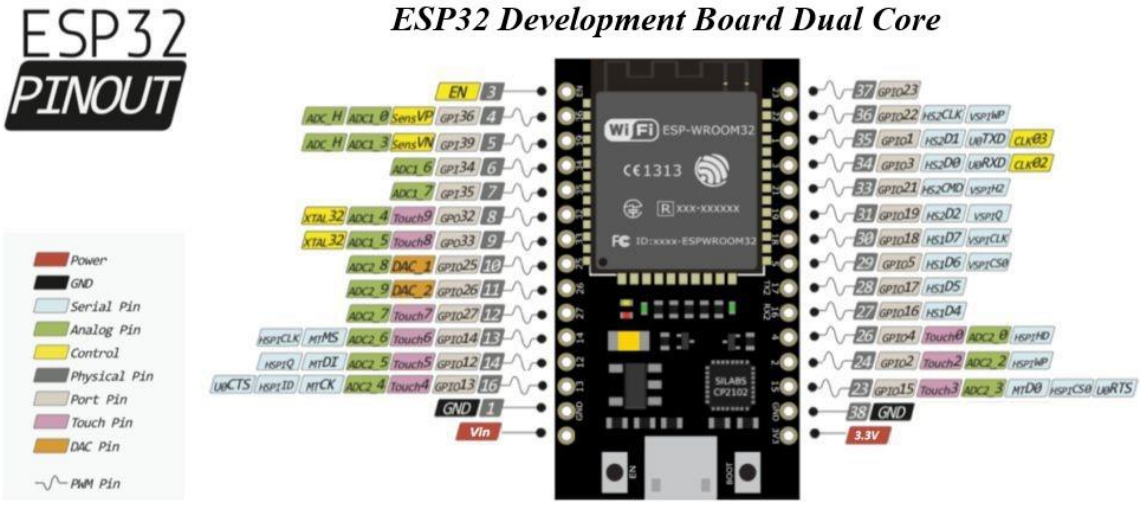


Figura 17: Disposición de los pines del ESP32

El punto negativo de estas mejoras lo encontramos, obviamente, en el **sobrecoste** que supone la adquisición de este chip con respecto al original *ESP8266*, puesto que las placas de desarrollo que integran el *ESP32* pueden encontrarse con un precio **entre 4 y 10 veces mayor** que aquellas con cualquier módulo del *ESP8266*. Podemos comprobar de un simple vistazo las diferencias técnicas entre estos dos chips con la siguiente tabla comparativa:

	ESP8266	***ESP32***
Procesador	Tensilica Xtensa LX106 x32 @80 MHz (hasta 160 MHz)	Tensilica Xtensa LX6 x32 @160 MHz (hasta 240 MHz)
CoProcesador	-	Si
Memoria RAM	96 kB (40 kB disponibles)	520 Kb
Memoria ROM	-	445 Kb
Flash	Hasta 4 MB	Hasta 16 MB
Alimentación	3.6 V	3.6 V
Consumo promedio	80 mA	80 mA
Consumo hibernación	20 µA (RTC + memoria RTC)	2.5 µA (10 µA RTC + memoria RTC)
WiFi	802.11 b/g/n (hasta +20 dBm) WEP, WPA	802.11 b/g/n (hasta +20 dBm) WEP, WPA
Bluetooth	-	V4.2 BR/EDR y BLE
GPIO	16	32
PWM	8	16

SPI	2	4
UART	2 (Una exclusiva para el pin Tx)	3
I2C	1	2
ADC	1 (10 bit)	18 (12 bit)
ADC con preamplificador	-	Sí (Bajo ruido) hasta 60 dB
DAC	-	2 (8 bits)
1-Wire	Por software	Por software
I2S	1	2
CAN bus	-	1 (2.0)
Ethernet	-	10/100 Mbps MAC
Sensor Temperatura	-	Sí
Sensor efecto Hall	-	Sí
IR	Sí	Sí
Temporizadores	3	4
Encriptación por hardware	-	Sí
Gen. números aleatorios	-	Sí
Encriptación Flash	-	Sí
Safe Boot	-	Sí

A pesar de las evidentes diferencias en cuanto a prestaciones que presentan el *ESP32* frente al *ESP8266*, este último **sigue siendo totalmente válido para muchos proyectos, siendo su principal atractivo su precio**, muy asequible para aquellos que quieren empezar a desarrollar prototipos de aplicaciones relacionadas con el IoT.

Estas son las placas de desarrollo que más se suelen utilizar a día de hoy, pero **existen otras muchas** que son igualmente interesantes y también son empleadas en diversos proyectos relacionados con el desarrollo de productos IoT, como por ejemplo las de *Libelium* (nacional), *BeagleBone*, *Intel Curie*, *NanoPi*, etc.

2.3. Tecnologías de comunicación

Otra de las tecnologías que están influyendo directamente sobre el desarrollo del Internet de las Cosas es sin duda las **comunicaciones**. Se pueden utilizar todos los estándares de infraestructuras de redes disponibles en la

actualidad en un proyecto IoT, tanto por cable como por radio. Sin embargo, los estándares de comunicaciones preferidos normalmente son los que emplean una **comunicación inalámbrica**.

Existen **diversos estándares** de comunicaciones inalambricas, dependiendo sobre todo del area de aplicación:

- Por un lado tenemos las **tecnologías tradicionales de conectividad inalambrica**, como el *WiFi* y la conectividad celular (*2G*, *3G* y *4G*), que son tecnologías de **alto consumo energético, pero ampliamente soportadas y cuentan con una gran cobertura**.
- Por otro lado, las **tecnologías de corto alcance**, que requieren el despliegue de repetidores o pasarelas. Estas han sido cl**aves para el éxito inicial del IoT, pero suelen ser inapropiadas en despliegues amplios**, pues se traslada la responsabilidad de operación de red al cliente final, algo que se ha comprobado es poco recomendable por los intentos iniciales realizados con estas tecnologías. Entre otras podemos destacar en este apartado a tecnologías como *ZigBree*, *Z-Wave*, *6LowPAN*, etc.
- Otro subgrupo de tecnologías de corto alcance son aquellas que ofrecen una **conectividad entre el objeto conectado y dispositivos móviles**. Estas son tecnologías como *Bluetooth*, especialmente con su última revisión de bajo consumo *BLE* (*Bluetooth Low Energy*), o la tecnología de comunicación sin contacto *NFC*.

Cabe destacar, además de las mencionadas anteriormente, una nueva corriente de tecnologías de comunicación para el IoT, **LPWAN** (*Low-Power Wide-Area Network*) que cuentan como características principales su **bajo consumo**, **largo alcance** y **bajo coste de dispositivos**, entre las que podemos destacar tres:

- **Sigfox**, un operador francés que se encuentra desplegando y gestionando su propia red basada en su propia tecnología, operado en España por *Cellnex Telecom* y que atesora el primer gran cliente *IoT* de nuestro país, *Securitas Direct*.

- **Lora**, alternativa a *Sigfox* que da la opción de desplegar redes privadas o bien ser usada por operadores para sus propias redes *IoT*. Ya se está desplegando en países como Francia o Alemania.
- **Evoluciones del *LTE/4G* adaptadas al *IoT***. Son la gran apuesta de las operadoras que se espera para este año 2018. Representan versiones reducidas del *4G*, optimizadas para aplicaciones de bajo consumo de datos y energía. Principalmente son las conocidas como *Cat M1* o *Cat M* y *Cat NB* o ***NB-IoT***.

Como se ha comentado, estas redes comparten la filosofía de estar diseñadas para consumos bajos, y la capacidad de ofrecer diseños de dispositivos con bajo coste, grandes coberturas y baja tasa de trasmisiones de datos, pero dependen de la madurez del despliegue de las redes, que **en estos momentos se encuentran en pleno proceso de transición y expansión**.

No obstante, hay que ver cómo afectará la implantación del futuro **5G**, que **también puede ser la solución para el IoT** en cuanto a las comunicaciones inalambricas se refiere para la gran mayoria de las aplicaciones, ya que según sus especificaciones permitirá un consumo eficiente, bajo coste y una gran cobertura, con lo que puede suceder que esta nueva tecnología desplace a las nuevas soluciones vistas anteriormente.

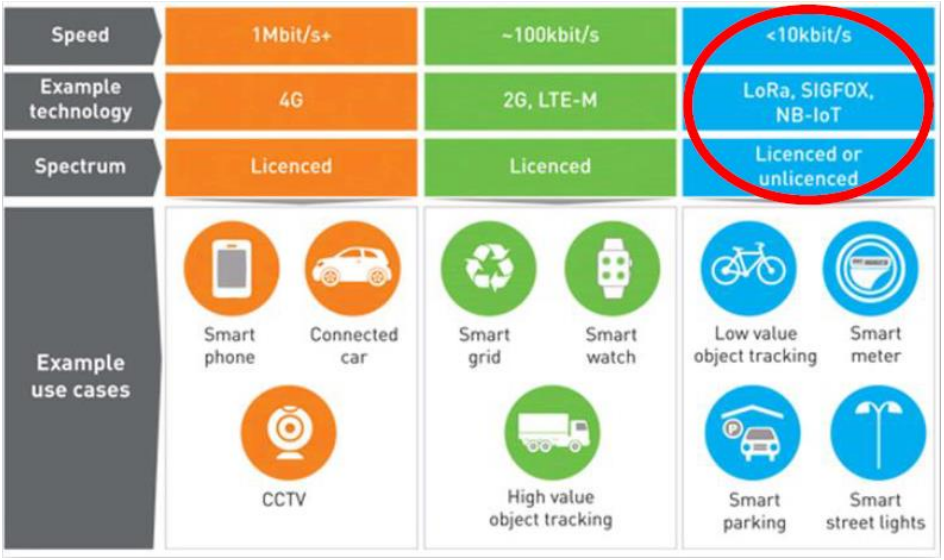


Figura 18: Redes inalámbricas celulares para IoT (Analysys Mason, 2015)

Hay que añadir también que aparte de estos estándares a nivel de infraestructuras de red, tambien existen **protocolos** que facilitan y simplifican el trabajo a los programadores de aplicaciones y proveedores de servicios para IoT, y que dependiendo de los tiempos de respuestas que se requieran y para que se vayan a utilizar será mejor usar unos u otros (ver figura 19). Por ejemplo, para la comunicación en tiempo real entre dispositivos en la industria se suele usar los protocolos *DDS* o *OPC UA*, para enviar datos desde los dispositivos a los servidores es preferible otros como *MQTT* o *CoAP*, para la comunicación entre los servidores es mas adecuado el protocolo *AMQP*, y para que los usuarios accedan a los datos en la nube es recomendable usar *XMPP* o *REST* (Schneider, 2013).

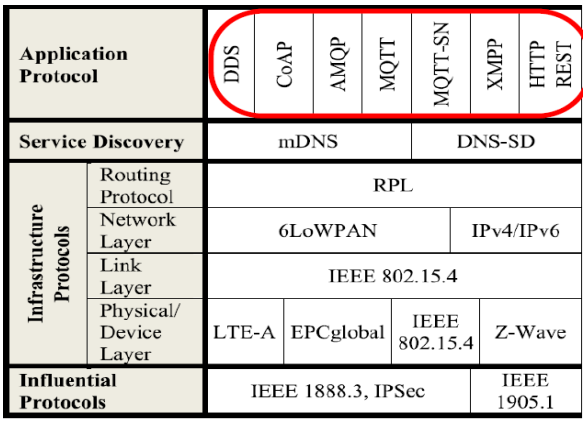


Figura 19: Protocolos de la capa de aplicación para el IoT (A. Al-Fuqaha, 2015)

2.4. Cloud Computing

Otra tecnología que está potenciando el desarrollo del Internet de las Cosas es la **computación en la nube** o Cloud Computing. Esta conclusión es obvia ya que a medida que existan más dispositivos conectados a Internet que incrementen la cantidad de datos que se transmiten (ver figura 20), se necesitará **más capacidad de procesamiento y memoria**, o lo que es lo mismo, se requiere que las infraestructuras puedan crecer, que sean **escalables**. Esta característica precisamente es la que nos otorgan soluciones cloud, como por ejemplo las de *Amazon Web Services*, *Microsoft Azure*, *OpenStack*, *OpenShift*, etc, que nos permitirán tener servicios que se ajusten a la demanda, un requerimiento clave en muchas de las aplicaciones IoT.

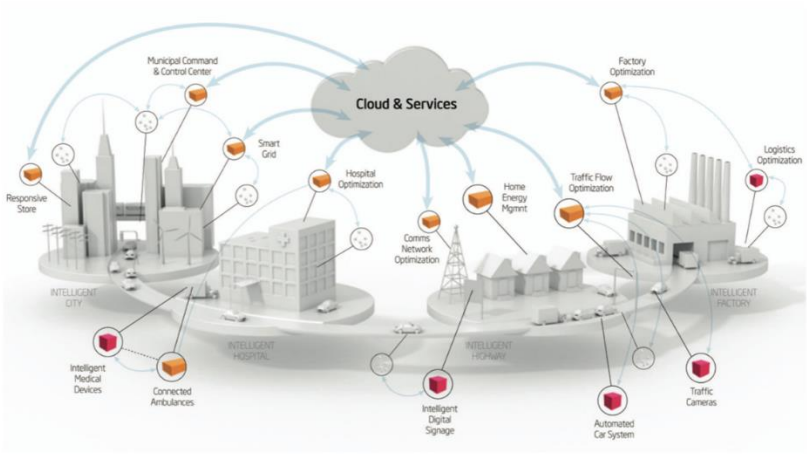


Figura 20: Esquema de funcionamiento del Cloud Computing

El problema del Cloud Computing es que no se aprovechan las capacidades de cómputo tanto de los millones de dispositivos como de los nodos y pasarelas de comunicación ya que todos los datos se envían a los grandes centros de datos donde son analizados y procesados.

Esto es precisamente lo que pretenden cambiar nuevas filosofías como el **Edge Computing**, en la cual se pretende que los datos recopilados por los dispositivos IoT se procesen más cerca de donde se crearon con el objetivo de, por un lado disminuir la cantidad de datos que se envían a la nube consumiendo **menos ancho de banda** (y como consecuencia **liberar la carga computacional de estos centros de datos**) y por otra parte conseguir **latencias más bajas** para datos importantes de manera que se puedan analizar y procesar en tiempo real, algo fundamental especialmente en escenarios empresariales e industriales, lo que se traduciría en una notable mejora en muchos procesos (Butler, 2017).

Por otra parte, otro término relacionado con el Edge Computing es el denominado **Fog Computing** que plantea extender la computación en la nube de manera que se acerque lo máximo posible a las “cosas” conectadas, afirmando que cualquier dispositivo con conectividad a la red y capacidad de almacenamiento y cómputo puede actuar como un nodo de esa “niebla” (Fog). De esta manera se pretende delegar parte del procesamiento de los grandes centros de datos que forman la nube a los **nodos de conmutación y las pasarelas de comunicaciones de manera que los cálculos se realicen más cerca de los dispositivos “edge”** (terminales, dispositivos IoT) (Cisco, 2015).

El ejemplo perfecto de caso de uso donde estos paradigmas pueden resultar de utilidad es el **coche autónomo**. En estos vehículos se genera una gran cantidad de datos que es necesario analizar y procesar en tiempo real para conseguir una conducción optima y segura. Por tanto es necesario que esto se realice en el centro de datos del propio vehículo ya que la **respuesta va a ser más rápida** que si tuviéramos que enviar los datos, realizar los cálculos en la nube y esperar la respuesta y a su vez estos datos "aprendidos" en cada uno de los eventos a los que se enfrente el vehículo puede **compartirse con la nube para que el resto de los modelos de vehículos autónomos se beneficien** de la experiencia de cada vehículo en particular.

2.5. Analítica de datos

Sin lugar a dudas una de las tecnologías que más han beneficiado el progreso del *IoT* de los últimos años es la **Ciencia de Datos**. Según las previsiones que hemos comentado al comienzo de este capítulo, para 2020 se estima que existan alrededor de 40 Zettabytes, y **gran parte de esos datos provendrán de los objetos conectados a Internet que autónomamente generarán información día tras día**. Por tanto, como hemos mencionado anteriormente todos esos datos recopilados gracias al *IoT* podrán convertirse en conocimiento mediante las técnicas y procedimientos adecuados que proporcionarán la *Ciencia de Datos*.

Es destacable sobre todo el estudio del ***Big Data*** para poder afrontar eficientemente esa ingente cantidad de datos heterogéneos, y los **algoritmos de minería de datos y de aprendizaje automático** para realizar predicciones, clasificaciones (por ejemplo para resolver problemas como los atascos de tráfico o gestionar el aparcamiento en grandes ciudades o la detección prematura de enfermedades cardiovasculares), dar recomendaciones, reconocer patrones o coincidencias (por ejemplo, para ayudar a los clientes de un supermercado a encontrar los productos que estén buscando), o tambien para descubrir comportamientos o detectar anomalías en el funcionamiento de máquinas industriales, entre otras muchas utilidades (R. Ciobanu, 2014). Todo ello

analizando los distintos datos que llegan desde los miles o incluso millones de sensores y dispositivos que se pueden tener en determinadas aplicaciones.

Mining Algorithm	Goal	Data Source
Clustering	Network performance enhancement	wireless sensor
	Inhabitant action prediction	X10 lamp and home appliances
	Provisioning of the needed services	Raw location tracking data
	Housekeeping	Vacuum sensor
	Managing the plant zones	GPS and sensor for agriculture
	Relationships in a social network	RFID, smart phone, PDA, and so on
Classification	Device recognition	RFID
	Traffic event detection	GPS, smart phone, and vehicle sensor
	Parking lot management	Passive infrared sensor
	Inhabitant action prediction	RFID, sensor, video camera, microphone, wearable kinematic sensor, and so on
	Inhabitant action prediction	Video camera
	Inhabitant action prediction	microphone
	physiology signal analysis	wireless ECG sensor
Frequent Pattern	RFID tag management	RFID
	Spatial colocation pattern analysis	GPS and sensor
	Purchase behavior analysis	RFID and sensor
	Inhabitant action prediction	RFID and sensor
Hybrid	Inhabitant action prediction	RFID and sensor

Figura 21: Algoritmos de minería de datos aplicados al IoT (Chun-Wei Tsai, 2014)

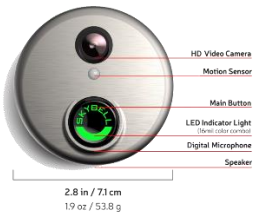
2.6. Ámbitos de aplicación del IoT

Al igual que Internet ha supuesto una tecnología que ha revolucionado casi cualquier ámbito de nuestra vida, el *IoT* es el siguiente paso que supondrá una mejora en todos aquellos ámbitos que directa o indirectamente se puedan beneficiar de esta tecnología para progresar y mejorar en calidad, eficiencia y costes (M. A. Razzaque, 2016). Los **ámbitos de aplicación relacionados con el IoT son muy diversos y heterogéneos**, sin embargo, aquí describiremos los que mayor evolución y protagonismo han alcanzado en estos últimos años:

2.6.1. Smart Home

El concepto de casa inteligente es, sin lugar a dudas, uno de los campos en los que el *IoT* más ha apostado en los ultimos años y en el que **más proyectos se llevan a cabo**, tanto por empresas como por universidades y particulares. Y es que este ambito se presta mucho a servirse de las utilidades que aporta el *IoT*, tales como el control de la temperatura de las distintas habitaciones de la vivienda, gestión remota de las puertas, persianas, etc, electrodomésticos inteligentes que nos avisan cuando precisan de nuestra atención o incluso que pueden llegar a hacer un pedido a la tienda si así lo deseamos. El abanico de

posibilidades es muy amplio. Algunos ejemplos concretos muy representativos son:

Nombre	Descripción	Enlace
Báscula Withings 	Báscula que te reconoce y envia tu peso y % de grasa corporal a tu smartphone.	http://www.withings.com/
Parrot Pot 	Macetero que integra distintos sensores para monitorizar el estado de las plantas mediante una aplicación movil.	https://www.parrot.com/es/jardin-conectado/parrot-pot
HapiFork 	Tenedor inteligente que ayuda a comer mejor.	https://www.hapi.com/product/hapifork
SkyBell 	Timbre WiFi, para responder de forma remota y visualizar en el movil quien está llamando a nuestra puerta.	http://www.skybell.com/
Bombilla LIFX 	Bombilla inteligente controlada por WiFi desde el teléfono movil.	http://lifx.co/

SmartThings 	Plataforma que permite añadir sensores de todo tipo a nuestro entorno doméstico.	http://www.smartthings.com/
Termostato Nest 	Termostato inteligente que se ajusta automaticamente para ahorrar energía en función de las preferencias del usuario.	https://nest.com/
Amazon Dash Button 	Dispositivo WiFi que te permite pedir un producto pulsando un botón.	http://www.amazon.es/dashbutton/

2.6.2. Smart City

El ámbito de las ciudades inteligentes es tambien un campo en el que las empresas e instituciones están apostando fuertemente. Es conveniente tener la mayor cantidad de datos posible para **facilitar la vida de los ciudadanos**. Obviamente este concepto es más util en las grandes ciudades ya que el volumen de datos que se puede conseguir es mayor y cuanto más datos se obtenga más y mejor conocimiento podremos obtener de ellos. Las aplicaciones abarcan desde el *Smart Parking*, indicando al conductor donde hay disponible un hueco para aparcar, el *Smart Traffic*, que calcula en tiempo real el tráfico en la ciudad y permite seleccionar rutas alternativas para nuestro viaje basandose en estos datos, hasta una gestión más eficiente del alumbrado de la ciudad.

Además gracias al IoT los **medios de transporte público evolucionarán hasta convertirse en los mejores sistemas para moverse por la ciudad**, pudiendo monitorizar en tiempo real la posicion de los vehiculos, ser alertados de imprevistos, predecir la llegada a nuestro destino con una precisión increible, trazar rutas alternativas personalizadas, obtener información sensible, avisar sobre averías de los vehículos, además de las aplicaciones que nos permitirán pagar más facilmente, organizar mejor los viajes y contaminar menos.

Aparte de esto tambien estamos viviendo los primeros intentos de crear **vehículos autonomos** que utilicen la información que nos transmite el Internet de las Cosas para circular libremente y de una manera más eficiente. De hecho, Elon Musk (*CEO* de *Tesla*) está proponiendo un modelo de negocio consistente en ofrecer un precio unico a la hora de comprar un *Tesla* que englobe al propio coche, el coste de asegurarlo y sus hipoteticos arreglos, ya que su creador está convencido de que tendrá menos averias y menos accidentes.

Nombre	Descripción	Enlace
Coche Tesla 	Vehículo autónomo que actualiza su software por Internet.	http://www.tesla.com/
Trackdot 	Objeto conectado que rastrea tus maletas cuando viajas.	http://www.trakdot.com/es/
Smart Electric Bike 	Bicicleta inteligente que ayuda a circular mejor por las ciudades.	https://www.smart.com/id/en/index/smart-electric-bike.html/

Estos son solo algunos ejemplos. **Su futuro es muy prometedor**, ya que existen diversas empresas que se están especializando y están apostando fuertemente por convertir las ciudades en un ecosistema más inteligente y conectado.

2.6.3. Agricultura inteligente

Otro de los ambitos donde el IoT está generando un gran beneficio es en el de la agricultura. Sus aplicaciones abarcan desde la monitorización de los cultivos, herramientas de soporte para la toma de decisiones, controlar automaticamente el riego y los niveles de los depósitos, detección y control de incendios forestales, protección de las heladas, fertilización, etc. La agricultura inteligente se convertirá en un futuro próximo en **una de las areas de aplicación más importantes en aquellos países predominantemente agrícolas**. Por ejemplo, en España existe una empresa llamada *Libelium* que está apostando fuerte por este ambito de aplicación. En la figura 22 se puede observar el Kit que ofrece dicha empresa para este ámbito de aplicación.

Figura 22: Kit IoT de Libelium para agricultura

2.6.4. Salud

El Internet de las Cosas va a permitir que se comercialicen nuevos sistemas de control biomedico que **controlan facilmente temas esenciales para nuestra salud** como nuestras constantes vitales, nuestra alimentación, el sueño, etc.

Hablamos, por ejemplo, de aplicaciones que controlan el correcto funcionamiento de un marcapasos y preveen situaciones que pueden resultar críticas para los pacientes.

En estos casos puede combinarse la recopilacion de información junto con la toma de decisiones posteriores: ropa interconectada, sistemas de control del ejercicio que realizamos mediante dispositivos wearables, etc. Algunos casos concretos son:

Nombre	Descripción	Enlace
Sen.se Mother 	Dispositivo que te cuida como una madre, ya que permite conectar muchos sensores que se harán cargo de su salud, seguridad y bienestar.	https://sen.se/store/mother/
Jawbone Up 	Pulsera de actividad que junto con los relojes inteligentes son de los dispositivos IoT de mayor éxito comercial actualmente.	https://jawbone.com/up
Shimmer 	Dispositivos wearables para la captura en tiempo real de señales para aplicaciones relacionadas con la salud y el bienestar.	http://www.shimmersensing.com

Como hemos visto en esta sección, las oportunidades de crear productos basados en el *IoT* **son casi infinitas** y estan generando nuevos modelos de negocio en diversos ambitos distintos entre sí, tanto para grandes como para pequeñas empresas (gracias a los bajos costes que caracterizan a los dispositivos conectados), que ven la oportunidad de invertir en campos como las *Smart Home* y *Smart Cities*, cuyas previsiones de crecimiento y beneficios son increíbles para los próximos años (W. Colitti, 2014).

Capítulo III

Plataformas IoT

En el capítulo anterior hemos expuesto muchos conceptos relacionados con el Internet de las Cosas tales como la propia definición del *IoT*, un breve repaso a su historia, las principales tecnologías habilitadoras o potenciadoras como las plataformas de desarrollo, tecnologías de comunicación, cloud computing o analítica de datos, además de los distintos ámbitos de aplicación donde más se están desarrollando aplicaciones relacionadas con el *IoT* y por supuesto la evolución que este concepto supone para el propio Internet.

Sin embargo no hemos mencionado a propósito unas **herramientas** que han surgido, como tantas otras tecnologías y modelos de negocio que hemos mencionado anteriormente, **gracias a la popularidad que está teniendo el Internet de las Cosas** en los últimos años y que, al igual que comentábamos sobre la relación entre el propio Internet y los servicios que le aportan la mayoría de sus utilidades, estas herramientas forman una **relación de beneficio recíproco con el Internet de las Cosas y son parte primordial en su éxito y popularidad** a día de hoy.

Estas herramientas a las que nos referimos son las denominadas **Plataformas IoT** y vamos a dedicarle integramente este capítulo para conocer mejor qué son, cómo funcionan, cómo se clasifican y cuales son aquellas más destacadas en la actualidad.

Además, como muestra de las ventajas que nos proporcionan las *Plataformas IoT* y para ilustrar de un modo más práctico sus utilidades y características

nosotros vamos a utilizar una de ellas para el desarrollo de nuestro prototipo *IoT* durante este trabajo.

3.1. Introducción

Vamos a empezar por los conceptos básicos. Definiremos qué es una *Plataforma IoT* y cuáles son las ventajas de utilizarlas, además de las características que las convierten en una opción muy recomendables a la hora de desarrollar soluciones *IoT*.

Debido a las complejidades que acompañan a todas las tecnologías que inciden sobre el IoT, surgió la necesidad de crear herramientas cuyo objetivo principal sea el de **proporcionar al desarrollador de todos aquellos aspectos técnicos necesarios**, es decir, las *Plataformas IoT* **facilitan el desarrollo integral de soluciones para el Internet de las Cosas**. Concretamente, las funcionalidades que suelen ser comúnes a todas las plataformas IoT son:

- Simplificar la **conectividad** y la normalización.
- **Gestionar los dispositivos** de forma centralizada.
- Añadir **reglas** para actuar y disparar eventos.
- Facilitar el **diseño de interfaces gráficas** de usuarios.
- **Almacenar los datos** recolectados por los dispositivos.
- **Análizar y procesar** dichos datos.
- Ofrecer interfaces (API, SDKs,…) para **integrarse con aplicaciones de terceros**.
- Ofrecer la **seguridad y privacidad** necesarias.

Podríamos definir las *Plataformas IoT* como unas herramientas cuya principal ventaja es la abstracción que proporcionan a la hora de desarrollar productos IoT, ya que las plataformas IoT están diseñadas de manera que los desarrolladores que las empleen en sus proyectos pueden **enfocar sus esfuerzos al desarrollo y optimización de los programas que ejecutarán las placas de desarrollo** (es decir, los dispositivos que conectarán con la plataforma IoT) **y a las aplicaciones que emplearán el usuario final** para gestionar y acceder a los datos proporcionados por nuestros dispositivos conectados (por

ejemplo una aplicación web o una aplicación de dispositivos móviles), dejando a un lado las complejidades técnicas que conllevarían tareas tales como la seguridad, la comunicación entre dispositivos o el almacenamiento, tratamiento y representación de los datos.

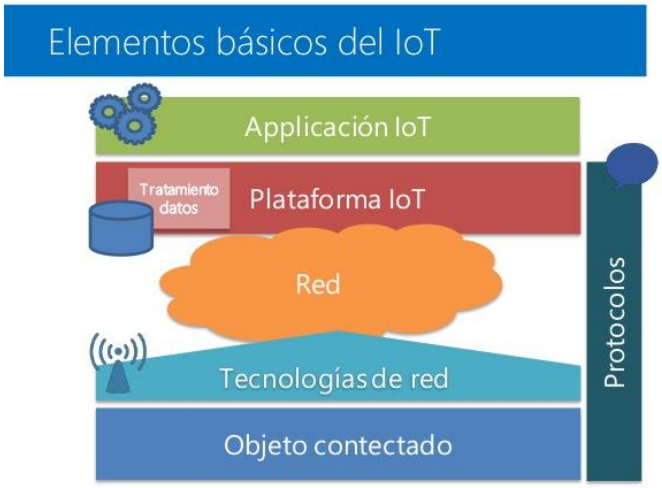


Figura 23: Ubicación de las Plataformas IoT

Otra definición dada por *Link labs*, una empresa estadounidense dedicada al desarrollo de tecnología de red para negocios e industria, para las plataformas IoT es “una plataforma web integrada que conecta hardware, puntos de acceso y redes de datos a lo que generalmente suele ser la aplicación de la que disfruta el usuario”.

Por tanto, todas y cada una de las funcionalidades que nos proporcionan las *Plataformas IoT* nos facilitan en gran medida la tarea de diseñar un producto *IoT*, más concretamente:

La **conectividad** y **normalización** que nos ofrecen garantiza una fiable, privada y segura conexión y comunicación con los diferentes dispositivos y usuarios, y con diferentes protocolos y formatos de datos.

La **gestión de los dispositivos** nos permite conocer a tiempo real el estado de cada uno de estos dispositivos y detectar fallos o averías tanto en los nodos como en la transmisión de los datos, además de añadir y/o eliminar dispositivos en cualquier momento.

El **almacenamiento** de los datos nos permite tener un sistema de base de datos completo y escalable para todos los datos que nos aportan los diferentes dispositivos, pudiendo aumentar el volumen de nuestra base de datos en cualquier momento, según las necesidades del usuario/cliente gracias a que las plataformas IoT son herramientas basadas en la nube. Además el **analisis** que podemos realizar sobre esos datos nos da la posibilidad de procesarlos, transformarlos y realizar multiples estadisticas que nos aporten una utilidad real a nuestras aplicaciones IoT.

Las diferentes **reglas de actuación y disparadores** que podremos configurar nos permitirán ejecutar acciones “inteligentes” basadas en los datos aportados por el sensor, como por ejemplo, encender automaticamente la calefacción de nuestro “hogar conectado” si detectamos que la temperatura de las habitaciones disminuye lo suficiente. Gracias a la analítica de datos podremos aplicar complejos algoritmos a nuestros datos y obtener conclusiones e incluso un aprendizaje automático por parte de ciertos dispositivos inteligentes conectados, es decir generar comportamientos basados en la información suministrada (Siguiendo con el ejemplo del “hogar conectado”, si tenemos configurada una hora para que se suban automaticamente las persianas de nuestro hogar al amanecer y llega el día que pasamos del horario de verano al de invierno, el sistema podria detectar que no es necesario subir las persianas a la hora que teniamos establecida, ya que al cambiar de horario todavia no habria salido el sol, y podria atrasar esa hora automaticamente).

Por otra parte, la capacidad de construir **interfaces gráficas** que suelen ofrecen las *Plataformas IoT* nos aporta una visión diferente de los datos recolectados, las estadisticas que podemos realizar y la evolución que sufren a lo largo del tiempo en el rango que más nos interese. Por ejemplo, podemos mostrar en un gráfico cómo ha evolucionado la temperatura de un lugar concreto, utilizando para ello un sensor de temperatura y un dispositivo conectado que se encarga de recoger y enviar los datos medidos a la plataforma IoT, durante las ultimas horas, días, semanas, etc.

Quizás uno de los aspectos más importantes de estas plataformas es la posibilidad de emplear **APIs y kits de desarrollo software (SDK)** para ofrecer

a los desarrolladores la capacidad de integrarlas en sistemas de terceros, lo que ofrece a estas herramientas una gran adaptabilidad, permitiendonos consultar, modificar y borrar la información almacenada en la *Plataforma IoT* desde otros dispositivos, como por ejemplo desde un smatphone con sistema operativo *Android*.

Cabe mencionar además que la mayoria de las *Plataformas IoT* ofrecen multitud de herramientas adicionales, especialmente pensadas para los desarrolladores, que facilitan las tareas de prototipar, testear, gestionar, controlar y comercializar sus productos *IoT*. La mayoria de estas herramientas son gratuitas con un volumen pequeño de tráfico de datos y dispositivos conectados, y ofrecen diferentes planes de pago para ir aumentando el nivel tanto de volumen de dispositivos como de datos almacenados, bajo un coste determinado para el desarrollador del producto *IoT* según el plan que más le interese en ese momento, pensados para aumentar el negocio de una manera progresiva y escalable.

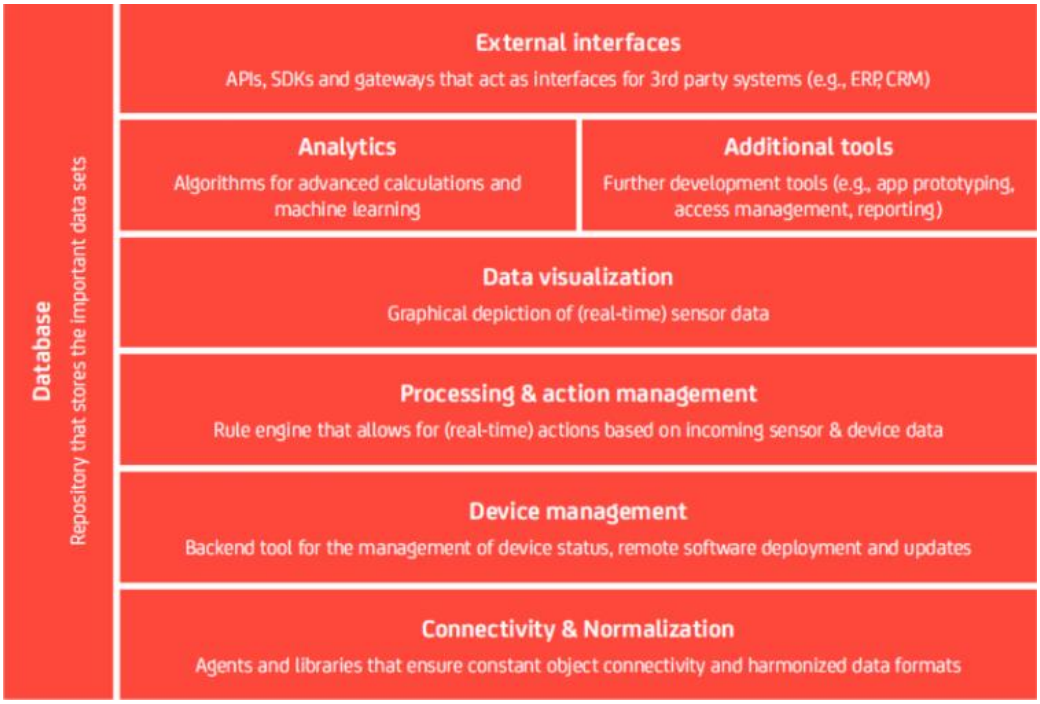


Figura 24: Funcionalidades de las Plataformas IoT (IoT Platforms: Market Report 2015-2021, 2016)

En la actualidad, debido al gran auge del Internet de las Cosas, existe una **gran proliferación de diferentes *Plataformas IoT*** que han surgido a lo largo de estos último años. De hecho, en la curva de *Gartner* que comentabamos en el capítulo

anterior, las *Plataformas IoT* se encuentran en plena fase de ascensión al pico de máxima expectativa y se encuentran entre las herramientas en las que se está trabajando e invirtiendo más actualmente. En junio de 2017 *IoT Analytics* dispuso una lista de 450 empresas de *Plataformas IoT*. Algunas de las ellas son *Open Source*, con lo que permiten utilizarlas y modificar su código fuente sin restricciones de licencia. Otras son propietarias pero ofrecen planes gratuitos para volúmenes bajos de dispositivos y/o comunicaciones, como hemos comentado anteriormente. Pese a ello, gracias a la gran adaptabilidad que ofrecen muchas veces este aspecto no suele suponer un problema incluso para los proyectos de productos *IoT* más exigentes. Por último podemos encontrar algunas *Plataformas* que ofertan un completo ecosistema IoT. Esto es, que la propia *Plataforma* nos ofrece el software, el hardware y todo lo necesario para desarrollar nuestra aplicación IoT.

A continuación vamos a llevar a cabo un **análisis** de algunas de las *Plataformas IoT* más representativas de los **distintos grupos que pueden existir actualmente**, para ver cuales son sus características y estudiar cuales son sus ventajas e inconvenientes con respecto al proyecto que deseemos llevar a cabo con ellas.

3.2. Clasificación de plataformas

Como ya hemos comentado anteriormente, el continuo avance en el Internet de las Cosas propicia a su vez el desarrollo de las Plataformas IoT, ya que son conceptos muy relacionados entre sí. Por tanto, cada día nacen nuevos proyectos IoT y nuevas Plataformas que suelen innovar en diversos aspectos y es complicado realizar un analisis en profundidad sobre ellas debido al gran volumen de que disponemos hoy en día.

En este capítulo vamos a intentar hacer una pequeña clasificación de Plataformas IoT atendiendo principalmente al **coste** que se requiere para desarrollar sobre ellas y al **sector** que van dirigidas. Vamos a diferenciar, teniendo en cuenta lo que hemos comentado, en cuatro grupos distintos y comentaremos algunas Plataformas pertenecientes a cada uno de estos grupos.

El primer grupo está compuesto por las plataformas de **código abierto**, es decir, todas aquellas que dan acceso al código sin restricciones.

El segundo grupo son las plataformas que permiten **su uso de una manera gratuita, con limitaciones** en cuanto al tráfico de mensajes que podemos recibir y el número de dispositivos conectados que podemos tener simultaneamente. Son por tanto Plataformas IoT orientadas principalmente a startups y empresas pequeñas, aunque también pueden emplearse con fines educativos.

El siguiente grupo de Plataformas IoT lo componen aquellas Plataformas que, si bien también ofrecen versiones de prueba gratuitas, ofrecen más servicios adicionales para formar un **ecosistema** IoT, con funcionalidades tales como almacenar webs, API para dispositivos moviles, bases de datos, etc.

Por último, el cuarto grupo engloba a aquellas Plataformas IoT **propietarias de grandes empresas**, que estan orientadas sobre todo al sector industrial y a grandes proyectos del IoT.

4.2.1. Plataformas IoT de código abierto

Este tipo de Plataformas se caracterizan por ser gratuitas y de codigo abierto (Open Source), por lo tanto no estan sujetas a licencias ni restricciones de Copyright lo cual nos otorga libertad a cualquier desarrollador para descargar su codigo fuente y acceder a su programación.

Por tanto estas Plataformas IoT son utiles para ser descargadas e instaladas de forma local en cualquier dispositivo compatible. Suelen estar basadas en servidores web como Apache o NodeJS por lo que requieren que el desarrollador tenga experiencia en programación del lado del servidor.

Por otro lado, al ser una herramienta de código abierto suele tener detrás una buena comunidad de desarrolladores que la utilizan y prestan ayuda a los nuevos usuarios.

Algunas de las Plataformas IoT pertenecientes a este grupo son **OpenHAB** (orientada a la automatización para el Smart Home basado en Java), **Node-RED**

(basada en NodeJS, contiene una buena herramienta gráfica para el desarrollo de productos IoT), **Kaa** (destaca por tener una gran compatibilidad ya que está disponible en casi todos los sistemas operativos y dispositivos, permite el desarrollo de aplicaciones tanto en el lado del servidor como del cliente), **OpenIoT** (orientada principalmente al ámbito de la Smart City, permite la conexión simultanea de multitud de sensores), **MajorDoMo** (diseñada para la automatización del hogar, gran compatibilidad y basado en la web) o **Zetta**, la cual vamos a describir a continuación, como ejemplo de Plataforma de código abierto y así comprobar cómo funcionan y cuales suelen ser las ventajas y casos de uso habituales de este grupo de Plataformas IoT.

Zetta es una Plataforma para el IoT de código abierto en la cual podemos construir APIs para la interacción de dispositivos conectados. Esta Plataforma IoT está basada en Node.js, un entorno en tiempo de ejecución multiplataforma también de código abierto para la parte del servidor. Por tanto para aquellos programadores familiarizados con Node.js se presenta como una Plataforma IoT muy fácil de usar, no siendo así para aquellos principiantes con Node.js que requeriran de un estudio y entrenamiento adicional para empezar a trabajar con ella.

Algunas de las características principales de esta Plataforma IoT son:

- Plataforma **Open Source**, sin ningun coste por su utilización.
- Actualmente tiene una pequeña **comunidad** de usuarios en constante crecimiento.
- **Basada en Node.js**, un entorno JavaScript del lado del servidor.
- Funciona en diversas plataformas distintas (**cross-platform**) y es facilmente desplegable en muchas plataformas basadas en la nube.
- Permite generar **APIs** para una comunicación cómoda y fiable entre dispositivos. Cada llamada está basada en la API por lo cual podemos

usarlas para cualquier proposito tales como enviar datos a otra plataforma de análisis para el procesamiento de los datos.

- Proporciona **Websockets** para transmitir eventos en tiempo real.
- Puede **soportar casi todos los protocolos** de dispositivos y convertirlos a HTML.

Permite crear aplicaciones utiles para conectar dispositivos, recibir peticiones e interactuar entre los distintos dispositivos. Y también programar nuestras propias peticiones y disparadores, por ejemplo para notificar cuando se conecta un nuevo dispositivo.

Muchos proyectos desarrollados por la comunidad con esta Plataforma IoT estan relacionados con el ambito del *Smart Home*, aunque existen ejemplos para otras aplicaciones:

- Home Security con BeagleBone: un prototipo para un sistema de seguridad simple conectado a internet utilizando para ello un microfono, un altavoz, un led y, la placa de desarrollo hardware BeagleBone Black.

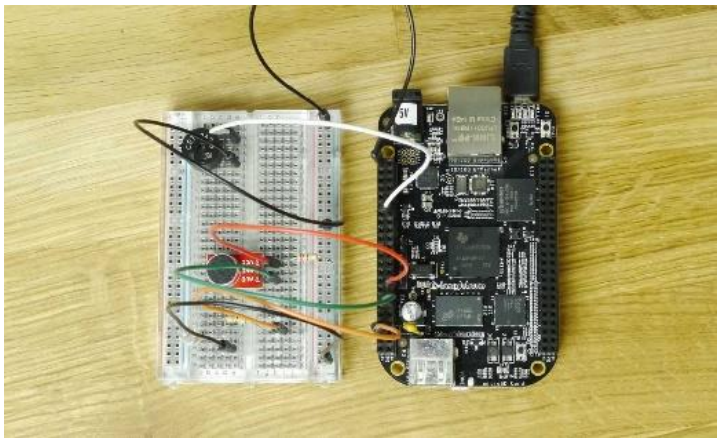


Figura 25: Sistema de seguridad desarrollado con Zetta y BeagleBone (www.zettas.org/projects)

- Home Security con Edison: proyecto similar al anterior pero utilizando como placa de desarrollo un Intel Edison.

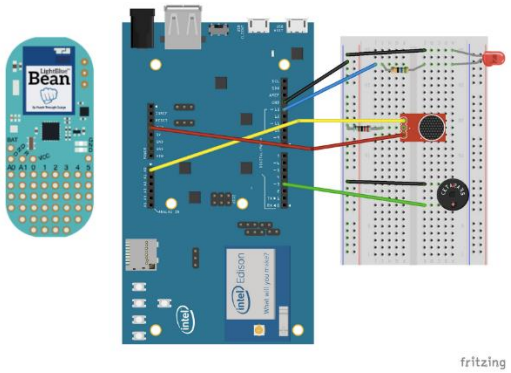


Figura 26: Sistema de seguridad desarrollado con Zetta e Intel Edison (www.zettajs.org/projects)

- Car Speed Tracker System: un prototipo de bombilla inteligente que se enciende cuando un vehículo supere los 100 km/h.

Figura 27: Detector de velocidad desarrollado con Zetta (www.zettajs.org/projects)

En conclusión, podemos decir que Zetta es una **plataforma gratuita enfocada principalmente al desarrollo de pequeñas aplicaciones y prototipos** que requieran la comunicación entre distintos dispositivos conectados. Si bien hay que destacar que **requiere un conocimiento previo de Node.js**, será una herramienta asequible para aquellos familiarizados con él, y su **capacidad para funcionar con un gran abanico de dispositivos** es una característica muy a tener en cuenta.

4.2.2. Plataformas IoT con versiones gratuitas

Este grupo estan formados por aquellas Plataformas IoT que disponen de versiones gratuitas, normalmente para un volumen pequeño de datos y dispositivos conectados, y de versiones de pago que permiten añadir muchos más dispositivos, almacenar más datos y poder generar un flujo mayor de estos datos hacia la Plataforma.

Son un grupo muy interesante porque normalmente las versiones gratuitas contienen todas las funcionalidades de las versiones mayores, lo que nos permite diseñar y desarrollar prototipos basandonos en dichas Plataformas IoT para despues, si la idea funciona, aumentar nuestras previsiones y poder llegar incluso a crear un modelo de negocio basado en nuestra nueva aplicación IoT. Además la mayoría de ellas suelen ofrecer facilidades para la conectividad de objetos y dispositivos de nuestros proyectos IoT con Arduino, una de las plataformas de desarrollo hardware más populares hoy en día, con todas sus ventajas añadidas.

Entre las más destacadas de este grupo se encuentran **Adafruit IO**, **PubNub**, **Ubidots**, **Thinger.io**, **Carriots** o **ThingSpeak**. Como hemos realizado en el anterior grupo, vamos a proceder a describir más detenidamente algunas de estas Plataformas.

ThingSpeak es una plataforma de aplicaciones diseñada para conectar personas con objetos. Nos facilita una **API** para almacenar y recuperar datos de los objetos utilizando para ello el protocolo HTTP desde Internet o desde una red de area local (LAN).

Es una plataforma basada en **Ruby on Rails 3.0 (RoR)**, un framework para aplicaciones web de código abierto basado en el lenguaje Ruby, con una arquitectura de Modelo-Vista-Controlador (MVC). Es un framework que aporta una gran simplicidad a la hora de programar, disminuyendo drásticamente el

código necesario y facilitando su configuración en comparación con otros framework, además de permitir el uso del “meta-programming” lo que da lugar a una sintaxis mucho más legible.

Esta herramienta nos ofrece todo lo necesario para poder desarrollar nuestros proyectos sobre el Internet de las Cosas, además de una aplicación web desde donde podemos gestionar todo lo relacionado con nuestra aplicación IoT, como los usuarios que están conectados, las claves de la API, etc. Sus características principales son:

- Dispone de una **API** que aporta al desarrollador los mecanismos necesarios para implementar su aplicación facilmente con la plataforma hardware deseada. Además la API de ThingSpeak está disponible desde GitHUB para su descarga a un servidor propio, y se ofrece completamente abierta con lo que podemos modificar su código fuente original. Esta característica permite que esta API esté en constante desarrollo por la comunidad, añadiendose nuevas caracteristicas cada cierto tiempo.
- La forma en la que ThingSpeak almacena los datos y los publica es a través de los denominados **canales**. Cada usuario puede crear los canales que estime oportunos de una manera muy simple y se pueden configurar como públicos o privados.
- La funcionalidad del sitio web de ThingSpeak se puede extender gracias a que se nos da la posibilidad de añadir **plugins** para crear aplicaciones de forma nativa en nuestra plataforma ThingSpeak. Soporta los lenguajes HTML, CSS y JavaScript. Al igual que ocurre con los canales, se pueden configurar como plugins públicos o privados según nuestros requisitos.
- Ofrece la posibilidad de emplear **Google Gauge Visualization**, lo que nos otorga la capacidad de visualizar facilmente los datos con un enorme nivel de personalización.
- Cuenta con una amplia capacidad de **integracion** con diversos dispositivos, concretamente podemos integrar ThingSpeak con:
    - Arduino
    - Raspberry Pi

  - IoBridge/RealTime.io
  - Electric Imp
  - Smartphones/Aplicaciones web
  - Redes Sociales

- Integra soporte para el popular software de computo numerico MATLAB permitiendo a los usuarios de esta Plataforma IoT realizar analisis, procesamiento y visualización en MATLAB de sus datos sin necesidad de disponer de una licencia de software. Además, toda la documentación de ThingSpeak está incorporada en el sitio web donde podemos encontrar la documentación de MATLAB e incluso podemos entrar en la web de ThingSpeak con nuestra cuenta de usuario de MathWorks.

Además de estas características ThingSpeak también ofrece a sus usuarios algunas aplicaciones disponibles desde su página web que pueden resultar de utilidad a la hora de desarrollar nuestros proyectos IoT. Actualmente en el catálogo disponible podemos encontrar:

- ThingTweet: permite a nuestro dispositivo publicar tweets en nuestra cuenta de la red social Twitter.
- ThingHTTP: permite conectar un dispositivo con microcontrolador a un servicio web. Soporta los métodos GET, POST, PUT y DELETE, ademas de SSL y autenticación.
- TweetControl: permite monitorizar hashtags de Twitter y conectar cualquier dispositivo a traves de ThingHTTP con esta red social.
- React: funciona como disparador que ejecuta una acción en cuanto se cumple alguna condición determinada con los datos de nuestros canales.
- TalkBack: Permite a un dispositivo actuar sobre los comandos en cola. Por ejemplo puede servirnos para programar el funcionamiento de una puerta para que se abra automaticamente si alguien se acerca y que despues de un par de minutos se cierre automaticamente si no detecta movimiento de personas.
- TimeControl: permite ejecutar a una determinada hora una funcion de ThingHTTP o de ThingTweet o añadir un nuevo comando a TalkBack.

Como podemos ver, nos encontramos ante una Plataforma IoT muy recomendada para todo aquel que se inice en el mundo del IoT. Su comunidad se compone en gran medida de usuarios que cumplen dicho perfil. Es muy util para prototipos aunque también puede aportarnos más funcionalidades si nuestro proyecto avanza lo suficiente y así lo requerimos. Sin embargo para un proyecto más profesional se va a requerir una **actualización a una cuenta de pago** ya que no nos serviría con el nivel básico gratuito, y los costes variarán dependiendo de la entidad y finalidad del proyecto a desarrollar.

ThingSpeak	FREE For small non-commercial projects and for evaluation of the service	STANDARD For all commercial, government and revenue generating activities	ACADEMIC For academic use by faculty, staff, or researchers at degree-granting institutions[1]	HOME For personal use only[1]
Scalable for larger projects	✖ No. Annual usage is capped.	✔	✔	✔
Number of messages	3 million/year (~8,200/day)[1]	33 million/year per unit (~90,000/day per unit)[1]	33 million/year per unit (~90,000/day per unit)[2]	33 million/year per unit (~90,000/day per unit)[2]
Message update interval limit	Every 15 seconds	Every second	Every second	Every second
MATLAB Compute Timeout	20 seconds	60 seconds	60 seconds	20 seconds
Number of simultaneous MQTT subscriptions	Limited to 3	50 per unit	50 per unit	50 per unit
Private channel sharing	Limited to 3 shares	Unlimited	Unlimited	Unlimited
Technical Support	Forum	Standard MathWorks support	Standard MathWorks support	Forum

Figura 28: Distintos planes de suscripción disponibles para ThingSpeak

Carriots es una **plataforma como servicio** (PaaS, de sus siglas en inglés) desarrollada en España para proyectos relacionados con el Internet de las Cosas y proyectos de Máquina a Máquina (M2M).

Es una Plataforma IoT **propietaria**. Podemos crear **una cuenta de forma totalmente gratuita y gestionar un máximo de 2 dispositivos**, aunque con algunas restricciones de funcionalidad. Sin embargo, hay que decir que para hacer un uso completo de la Plataforma unicamente hay que pagar un coste por dispositivo extra dependiendo de la cantidad de dispositivos y el tráfico de datos que estos generen, con un precio para nada desorbitado en comparación con otras plataformas similares.

FREE	CORPORATE	LITE	PRIVATE CLOUD ON PREMISE
Free (NO CREDIT CARD REQUIRED)	2 €* PER MONTH PER DEVICE	0,50 €* PER MONTH PER DEVICE	Contact Us
FOR TESTING AND PROTOTYPING	DEVICES SENDING UP TO 1 MB PER DAY AND MAKING LOTS OF CONNECTIONS	LOTS OF DEVICES SENDING LOW AMOUNT OF DATA. E.G. SMART METER, REMOTE MAINTENANCE	FOR UNLIMITED CONNECTIONS AND DATA STORAGE OR PRIVATE USAGE OR CUSTOM NEEDS

Figura 29: Planes de suscripción de Carriots

Ofrece una **gran compatibilidad** con dispositivos y funciona mediante una **API** que utiliza el protocolo REST para transmitir datos. Además también ofrece un completo kit de desarrollo SDK basado en Java. Por tanto, se compone de un entorno de desarrollo sencillo, completas APIs y alojamiento de información que nos proporciona una situación ideal para el desarrollo de nuestras aplicaciones basadas en el IoT o en M2M y una gran **escalabilidad** ya que se pueden contratar los recursos necesarios a medida que la embergadura de nuestros proyectos lo requiera. Entre sus principales características se encuentran:

- **Gestión de dispositivos**: Monitorización y control sobre todos nuestros dispositivos conectados. Permite controlar su estado, cambiar sus configuraciones, activarlos o desactivarlos y actualizar su firmware.
- ***Listeners***: Se pueden crear sencillas estructuras IF-THEN-ELSE para realizar acciones cuando se reciba un dato, se almacene, se conecte un dispositivo, etc.
- **Reglas**: Podemos añadir lógica a nuestras aplicaciones IoT mediante scripts de código Groovy almacenados la propia Plataforma IoT. Listos para ser activados en los escuchadores de eventos o “listeners”.
- **Exportación de datos**: Permite transferir nuestros datos a un sistema externo a traves de los disparadores de Carriots. Podemos crear nuestros archivos de exportación, transferir datos a otras bases de datos o utilizar la API para gestionar el envío de nuestros datos.
- **Motor de aplicaciones SDK**: Los disparadores y las reglas son ejecutados por el motor SDK de Carriots. Flujos de ejecución de Java con aislamiento de procesos para garantizar la seguridad y la eficiencia.

- **Sistema de alarmas a medida**: Carriots proporciona una serie de alarmas automáticas cuando aparece algún error inesperado, pero también se pueden crear y gestionar nuestras propias alarmas.
- **Depuración y trazas**: Proporciona una consola para conocer cómo interaccionan los dispositivos con la plataforma.
- **Jerarquía de proyectos**: Permite albergar varios proyectos en una misma cuenta, para poder operar con distintos clientes y distintas visibilidades. Cuenta con una jerarquía de 7 niveles, desde el cliente hasta el dispositivo. Esta caracteristica es exclusiva de las cuentas de pago, no estando por tanto disponible para las cuentas gratuitas.
- **Gestión de claves API**: Utilidad para modificar los permisos y la visibilidad de los dispositivos y clientes que tengamos asociados. Opción disponible solo si disponemos de una cuenta de pago.
- **Gestión de usuarios**: Se pueden establecer usuarios corporativos que pueden administrar otras cuentas de usuario para ayudar a gestionar proyectos o para definir los permisos y visibilidad a la medida del cliente. Esta opción solo está disponible en cuentas de pago.
- **Personalización**: Podemos crear nuestro panel de control a medida y gestionar todas las caracteristicas de Carriots mediante la API.

Como ya hemos comentado, Carriots ofrece compatibilidad con un gran número de dispositivos hardware disponible en el mercado actualmente: **Arduino**, **Raspberry Pi**, Beagle Bone, Fez Cerbuino, Cubie Board, TST Gate, TST Mote, CloudGate, Nanode, Electric IMP, TST Light y **cualquier otro hardware que pueda conectarse a Internet y comunicarse mediante su API REST**.

Estamos ante una Plataforma IoT muy completa que nos ofrece las **funcionalidades necesarias** para realizar proyectos IoT en cualquier ambito de aplicación. En algunos de los más importantes tales como Smart Cities, Smart Energy, Smart Agriculture, Smart Building, Smart Retail y Smart Logistic ya se han desarrollado y comercializado aplicaciones, como por ejemplo el **proyecto de Smart City realizado en colaboración con la empresa Orange en la localidad de Pozuelo de Alarcón**, con el objetivo de disminuir la contaminación y proteger el medio ambiente.

En conclusión, podemos decir que Carriots ofrece una **gran variedad de soluciones de negocio para el IoT en distintos ambitos de aplicación**. Sus principales ventajas son la de contar con una API basada en un protocolo de comunicación ampliamente reconocido y utilizado como es REST muy bien documentada, gran compatibilidad con las plataformas de desarrollo hardware disponibles en el mercado y una gran versatilidad y escalabilidad. Es por tanto una **opción ideal para proyectos reales IoT**, aunque su cuenta gratuita también nos permite experimentar y desarrollar pequeños prototipos con la plataforma.

4.2.3. Plataformas IoT con integración completa

En este grupo vamos a encontrar aquellas Plataformas IoT que ofrecen mayores prestaciones, normalmente ofreciendo un software y un hardware específico para trabajar en la plataforma, y que suelen aportar más funcionalidades aparte de las típicas de almacenamiento y transferencia de datos, como son por ejemplo una mayor rapidez de implementación y mayor seguridad.

La mayoria de estas Plataformas IoT no cuentan con cuentas gratuitas pero sí suelen ofrecer la posibilidad de probar versiones de prueba durante un periodo limitado de tiempo. Como ejemplos de este grupo de Plataformas podemos nombrar **Electric Imp**, **TheThingsIO** o **Particle**.

Esta plataforma propone tanto soluciones hardware como soluciones software en forma de servicios. Su sello distintivo es disponer de soluciones finales con total integración incluyendo dispositivo hardware, APIs, sistemas operativos, servicios en la nube, seguridad y eficiencia, reduciendo los costes totales de manera significativa. Podriamos definirla como una plataforma IoT todo en uno.

Está basada en un módulo WiFi que comercializan, el cual tiene compatibilidad con cualquier dispositivo hardware y actúa como un gateway entre el dispositivo hardware específico que se utilice e Internet.

Es una plataforma que nos ofrece un ecosistema cuyas caracteristicas más importantes y diferenciadoras son:

- **ImpOS es el core** de Electronic Imp, que a traves del dispositivo y su agente online proporciona las herramientas necesarias para desarrollar cualquier aplicación. El código de nuestra aplicación correrá en una máquina virtual que nos ofrece ImpOS.
- Dispone de una **API** abierta que permite la personalizacion de la plataforma para adaptarla según nuestros requisitos. Basada en el lenguaje Squirrel, la API nos da opciones para gestionar la conectividad, energía, seguridad, notificaciones, etc.
- El **servicio impCloud** nos ayuda a conectar nuestros dispositivos a través de Internet asociandole a cada uno un único **agente** online para tener así total libertad a la hora de programarlo. Las principales funcionalidades que se nos ofrecen son el almacenamiento de datos y visualización a través del agente que se encarga de gestionar las comunicaciones bajo el protocolo HTTP.
- Dispone de una herramienta para smartphones, denominada **BlinkUp**, que funciona con dispositivos Android e iOS. Su función es la de configurar facilmente la conexión de nuestros dispositivos a Internet. Funciona transmitiendo la información WiFi de que dispone el smartphone al modulo Imp que dispongamos mediante una secuencia de flashes generada en la pantalla de nuestro smartphone, por lo que simplemente tendriamos que acercar físicamente la pantalla de nuestro smartphone al modulo Imp para configurar nuestro dispositivo hardware con el WiFi al cual esté conectado nuestro smartphone. Podemos ver un ejemplo de su uso en el siguiente enlace https://electricimp.com/platform/blinkup/

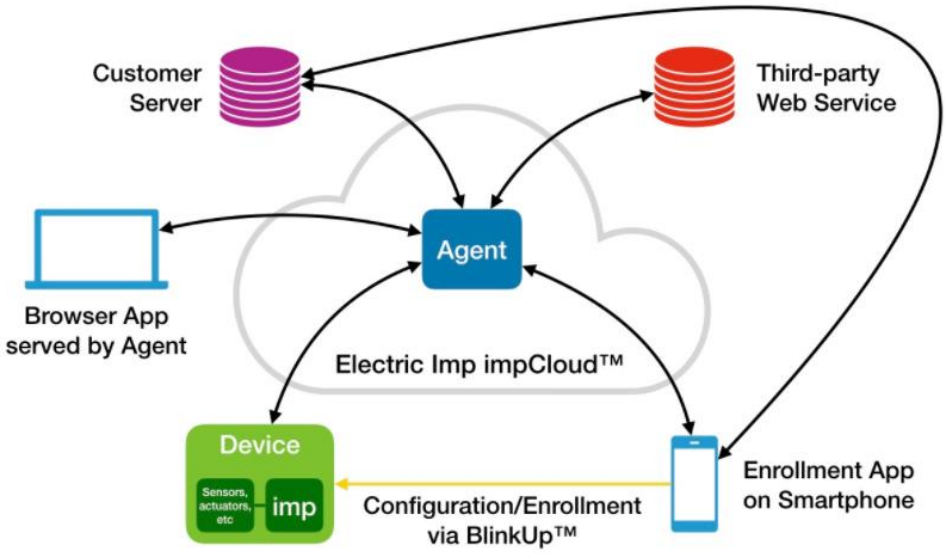


Figura 30: Esquema de funcionamiento de Electric Imp

Como ya hemos comentado, el éxito de Electronic Imp se encuentra en los módulos WiFi que distribuye y permiten conectar cualquier dispositivo hardware a Internet. Actualmente en el mercado dispone de tres módulos:

- Imp003: modulo compacto y con disponibilidad de modos de bajo consumo, además de una considerable capacidad y velocidad de memoria integrada.
- Imp004m: modulo más integrado que el anterior, con WiFi de 2.4 GHz.
- Imp005: modulo disponible con mayores especificaciones, ya que cuenta con WiFi doble banda (2.4/2.5 GHz) y Ethernet a 10/100Mb/s.

Electric Imp ofrece una **solución extremo a extremo** cuya principal ventaja es la abstracción, haciendo posible conectar cualquier dispositivo de una forma sencilla y rápida, dejando los esfuerzos del desarrollador al completo centrados en la aplicación IoT.

Particle es una plataforma IoT "All-In-One" (Todo en Uno) que solo permite su uso con los dispositivos hardware que ellos fabrican, la placa de desarrollo **Photon**, disponible a partir de 19$. El principal objetivo de esta Plataforma es aportar al cliente lo necesario para conseguir desarrollar y desplegar rápidamente y de una manera cómoda sus prototipos para, más tarde y si se desea, poder aumentar el nivel del proyecto a partir de la base que ya se ha conseguido. Entre sus características podemos encontrar:

- Permite **establecer conexión con los dispositivos fácilmente** ya que todo el hardware que distribuyen está diseñado para poder conectarse a la Plataforma IoT en minutos gracias a una **API**.
- Las placas de desarrollo de Particle funcionan con un sistema operativo propio, denominado **Device OS**, el cual es muy ligero y está diseñado específicamente para dicho hardware.
- Todos los dispositivos Particle se conectan mediante una API a **Device Cloud**, la plataforma web fiable, segura y escalable a través de la cual los clientes pueden gestionar, controlar y programar online sus dispositivos mediante **herramientas de administración y un entorno de desarrollo web** (**Web IDE**)**.** Además, también cuenta con otro entorno de desarrollo de escritorio (**Desktop IDE**) con más características y funciones.
- La **seguridad** en Particle es una característica muy importante. Cada mensaje intercambiado entre un dispositivo Particle y la Plataforma IoT se transmite **encriptado**, no estando permitido la transmisión de datos en texto plano. Además, cada dispositivo contiene su propia **clave privada**, evitándose así que un dispositivo no autorizado se conecte camuflándose entre nuestros dispositivos.

4.2.4. Plataformas IoT propietarias de grandes empresas

En este último grupo encontramos a aquellas Plataformas IoT que son propiedad de grandes empresas del sector de la tecnología, tales como Microsoft, Google o IBM, aunque también hay empresas en otros sectores que también tienen su propia Plataforma IoT propietaria, como es el caso de Amazon.

Casi todas estas Plataformas requieren de pago para su utilización, si bien suelen ofrecer una versión de prueba durante un periodo de tiempo limitado. Por tanto estan muy orientadas tanto para uso profesional como para el sector industrial, gestión de maquinaria y control de dispositivos a gran escala.

Para nombrar algunos ejemplos, como hemos mencionado, sólo tenemos que recurrir a grandes empresas del sector de las TIC o grandes multinacionales de otros sectores, tales como **Google Cloud Platform**, **Samsung Artic Cloud**, **IBM Watson IoT**, **Microsoft Azure** o **Amazon Web Services IoT**. Para terminar vamos a describir algunas de las más representativas de este grupo.

El servicio ofrecido por Google está diseñado para librar a los desarrolladores de tareas como la administración de la infraestructura, el aprovisionamiento de servidores y la configuración de redes. Se constituye por una **suite de servicios de computación basados en la nube** que permiten ejecutar nuestras aplicaciones con la misma tecnología y herramientas que se utilizan en Google, dando solución a todas las necesidades que un proyecto IoT a gran escala pueda requerir:

- **Recursos informáticos**
    - Compute Engine: Despliega máquinas virtuales escalables de alto rendimiento en los innovadores centros de datos de Google. Permiten escalar desde instancias individuales hasta un entorno de Cloud Computing global.

- App Engine: Plataforma como servicio (PaaS) para crear aplicaciones web y backends móviles escalables.
- Container Engine: Herramienta de administración y organización de clústers[14] que ejecuta contenedores Docker[15] en Google Cloud Platform. Permite programar los contenedores en el clúster y los administra automáticmente en función de los requisitos definidos (CPU, memoria, etc).

- **Almacenamiento y bases de datos**

  - Cloud Storage: sistema de almacenamiento de objetos unificado, que abarca desde el suministro de datos activos hasta el aprendizaje automático y el análisis de datos, pasando por el archivado.
  - Cloud SQL: Servicio que facilita la configuración, mantenimiento y la administración de las bases de datos MySQL relacionales en la nube. Ofrece altos niveles de rendimiento, escalabilidad y comodidad.
  - Cloud Bigtable: Servicio de base de datos de Big Data NoSQL de Google. Se trata de la misma base de datos que utiliza mucho de los servicios de Google, como Búsqueda, Analytics, Maps o Gmail. Está diseñado para administrar grandes cargas de trabajo con baja latencia y alto rendimiento uniformes, asi que es la elección ideal para el Internet de las Cosas.

- **Redes**

  - Virtual Private Cloud (VPC): Permite conectar nuestros recursos entre sí y aislarlos unos de otros dentro de una nube privada virtual.
  - Cloud Load Balancing: Herramienta de balanceo de carga escalable de alto rendimiento. Permite escalar nuestras aplicaciones de Compute Engine desde cero hasta pleno rendimiento sin necesidad de prepararlas antes, gracias a una escalabilidad automática inteligente.

[14] Conjunto de computadoras diseñadas con hardware comunes y que se comportan como una única computadora.
[15] Proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción.

  - Cloud DNS: Servicio de sistemas de nombres de dominio (DNS) caracterizado por su baja latencia y alta disponibilidad.

- **Big Data**

  - Big Query: almacenamiento de datos de gran escala totalmente administrado.
  - Cloud DataFlow: servicio de procesamiento de datos administrado capaz de ejecutar canalizaciones de datos continuas y por lotes. Capaz de ejecutar una gran variedad de patrones de procesamiento.
  - Cloud DataLab: herramienta interactiva que permite visualizar y analizar datos, así como desarrollar modelos de aprendizaje automático.

- **Cloud IoT Core**: Un completo servicio administrado para conectar, gestionar y obtener datos facilmente y de forma segura de nuestros dispositivos conectados.
- **Cloud Machine Learning Engine**: servicio para desarrollar fácilmente modelos de aprendizaje automático.
- **Identidad y seguridad**

  - Cloud IAM: herramienta para gestionar la visibilidad, el control de acceso y adminstrar de forma centralizada nuestros recursos en la nube.
  - Security Key Enforcement: protección ampliada contra la suplantación de identidad.

- **Herramientas de administración:**

  - Stackdriver Monitoring: permite monitorizar el rendimiento, tiempo de funcionamiento y estado general de las aplicaciones en la nube. Permite identificar tendencias y anticiparse a la hora de resolver problemas con nuestras aplicaciones y dispositivos.
  - Stackdriver Logging: administración y análisis de registros en tiempo real.

- Stackdriver Debugger: herramienta que permite inspeccionar el estado de una aplicación sin usar sentencias de registro ni detenerla o ralentizarla.

Es una Plataforma muy utilizada para los negocios por parte de grandes empresas como HTC, Spotify, Coca Cola, Motorola o Evernote. Entre las ventajas que ofrece esta Plataforma IoT podemos destacar:

- **Infraestructura** muy sólida, segura y en constante evolución. La nube de Google está diseñada para no sufrir fallos y funcionar a largo plazo.
- Fuerte apuesta por el **Big Data**, destacando el interes de la Plataforma en mejorar su facilidad de uso para los clientes y demostrando los beneficios que la Ciencia de datos es capaz de ofrecer.
- **Recursos informáticos sin servidor y totalmente administrados**, lo que permite a los clientes pasar rápidamente del prototipo a la producción y posteriormente a aumentar el nivel de negocio sin preocupaciones.
- **Política de precios innovadora** que apuesta por la facturación por minuto y los descuentos por uso continuado, lo que según Google se traduce en un ahorro en costes para el cliente.
- Modelo de **seguridad** arropado por los 15 años que la empresa lleva protegiendo los datos de sus clientes en sus aplicaciones más populares (Gmail, Search, Maps, etc).

Ofrece un **kit de desarrollo** con una placa BeagleBone como plataforma hardware especialmente adapatada para su uso con Cloud Platform.

Permite un periodo gratuito de 60 días, aunque requiere que registremos una tarjeta de crédito o cuenta bancaria válidas y cuando finaliza este periodo **la facturación depende de los servicios empleados y ofrece descuentos por uso continuado**.

IBM Watson IoT Platform

Otra gran empresa de la industria como es IBM apuesta fuerte por el IoT ofreciendo una amplia gama de servicios en su Plataforma IoT. Sus servicios son catalogados por áreas específicas tales como automoviles, hogar, electrónica o seguridad.

Proporciona un **potente panel de control web** flexible, escalable y fácil de utilizar. Con una interfaz de usuario sencilla donde poder añadir y gestionar nuestros dispositivos, controlar el acceso al servicio y supervisar su uso. Entre sus características se encuentran:

- **Gestión de dispositivos**: permite llevar a cabo acciones como reiniciar un dispositivo, actualizar su software, recibir metadatos, obtener un diagnostico de su funcionamiento y realizar acciones en masa con un gran número de dispositivos simultaneamente.
- **Conectividad fiable y escalable**: utiliza el protocolo MQTT para establecer conexiones entre los dispositivos y aplicaciones. Dicho protocolo es considerado un estandar del sector del IoT ya que está diseñado para realizar transmisiones eficientes desde y hacia los dispositivos conectados en tiempo real.
- **Comunicación segura**: gracias igualmente al protocolo MQTT con TLS (seguridad en la capa de transporte) para proteger las comunicaciones entre dispositivos y el servicio web.
- **Almacenamiento y acceso de datos**: se puede acceder a los datos arrojados por los dispositivos a tiempo real o también almacenarlos durante un periodo de tiempo a elección del cliente.
- **Compatible con IBM Bluemix**: en combinación con la plataforma IBM Bluemix, Watson IoT Platform proporciona un acceso simple pero potente a los datos almacenados y nuestros dispositivos IoT. Además podemos emplear aplicaciones analíticas, configurar paneles de control visuales y aplicaciones móviles de IoT.

Al igual que la Plataforma IoT anterior, permite un periodo de prueba limitado de 30 días y después **requiere de una suscripción** que variará en coste dependiendo de los servicios consumidos.

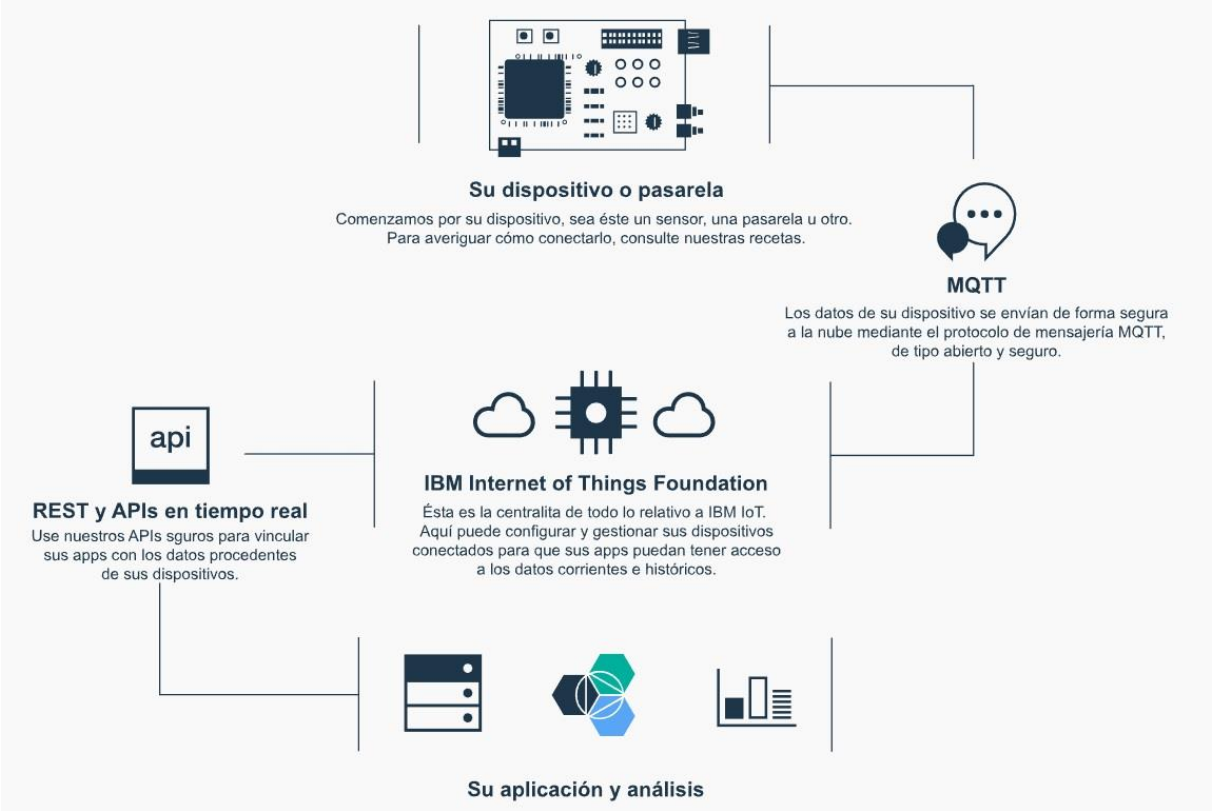


Figura 31: Esquema de funcionamiento de Watson IoT Platform

3.3. Evaluación y comparativa

A modo de resumen y evaluación final vamos a realizar una pequeña comparativa de las características más destacables que puede poseer una Plataforma IoT entre los distintos ejemplos que hemos mencionado en el apartado anterior, con el objetivo de poder comparar a simple vista las ventajas e inconvenientes de emplear una u otra para un proyecto IoT.

Plataforma	Hardware	Ámbito	Ventajas	Desventajas
Zetta	- Gran abanico de plataformas de desarrollo hardware	- Prototipos - Pequeñas aplicaciones	- Open Source - Cross-Platform - Gran compatibilidad con protocolos de comunicación	- Requiere conocimiento de Node.js - Poca documentación
ThingSpeak	- Arduino - Spark - Raspberry Pi Entre otras	- Smart Home - Prototipos	- Fácil de utilizar - Integración con redes sociales - Gratuita en pequeña escala, pero escalable - Funciones de MATLAB	- Documentación limitada a cierto Hardware
Carriots	- Arduino - Raspberry Pi - BeagleBone - TST - Industrial Cualquier hardware que pueda conectarse a Internet y usar su API REST	- Smart City - Smart Energy - Smart Agriculture - Smart Retail - Smart Logistic Entre otros	- Integración con redes sociales - API que emplea protocolo REST - Gran cantidad de hardware compatible - Útil en casi cualquier ámbito de aplicación - Buena escalabilidad con un coste moderado	- Poca documentación
Electric Imp	- Propietario	- Industrial - Smart Logistic - Smart Consumer - Smart Home	- Hardware dedicado - Ecosistema propio - Escalable	- No recomendable para iniciarse en el IoT
Particle	- Propietario	- Prototipos - Cualquier ámbito del IoT	- Facilidad para pasar del prototipo a la producción - La conectividad entre los dispositivos y la plataforma es trivial - Seguridad de las comunicaciones	- Solo permite su uso con el hardware propietario
Google Cloud Platform	- Cualquier plataforma de desarrollo	Cualquier ámbito relacionado con el IoT, principalmente Industrial	- Gran capacidad de escalamiento - Uso de las infraestructuras de Google - Suite de servicios muy variada - Kit de desarrollo propio con BeagleBone	- Requiere suscripción de pago
IBM Watson IoT Platform	- Cualquier plataforma de desarrollo	Cualquier ámbito relacionado con el IoT, principalmente Industrial	- Potente y sencillo panel de control web - Comunicaciones mediante protocolo MQTT con TLS - Compatible con IBM Bluemix	- Requiere suscripción de pago

Capítulo IV

Análisis del Producto IoT

En nuestro proyecto, además de conocer los conceptos relacionados con el Internet de las Cosas que hemos visto anteriormente, vamos a desarrollar un prototipo de producto *IoT* ya que es la manera más ilustrativa de mostrar cómo se desarrolla un producto de este tipo. Para ello primero vamos a poner en perspectiva qué tipo de producto queremos desarrollar y cuales son los requisitos que nuestro prototipo va a implementar.

4.1. Descripción de la aplicación IoT

Nuestro prototipo consistirá en una placa de desarrollo hardware que funcionará como un dispositivo que detecte el volumen de fluido que presenta un determinado contenedor, el cual puede ser un depósito de agua para el riego agrícola, un depósito de combustible para una caldera de un hogar, o cualquier otro recipiente cuya función sea contener algún tipo de fluido.

El principal propósito del prototipo será controlar cómo evoluciona el nivel del fluido dentro del depósito. Para ello vamos a necesitar **una placa de desarrollo hardware** que ejecute el programa que desarrollaremos para obtener el dato del sensor y conseguir la comunicación con una Plataforma IoT o una aplicación propia, además de un **sensor ultrasónico** que nos devolverá un dato de distancia (desde el sensor hasta el fluido) que utilizaremos para obtener el

volumen de líquido presente en el depósito, como se puede ver en la siguiente figura:

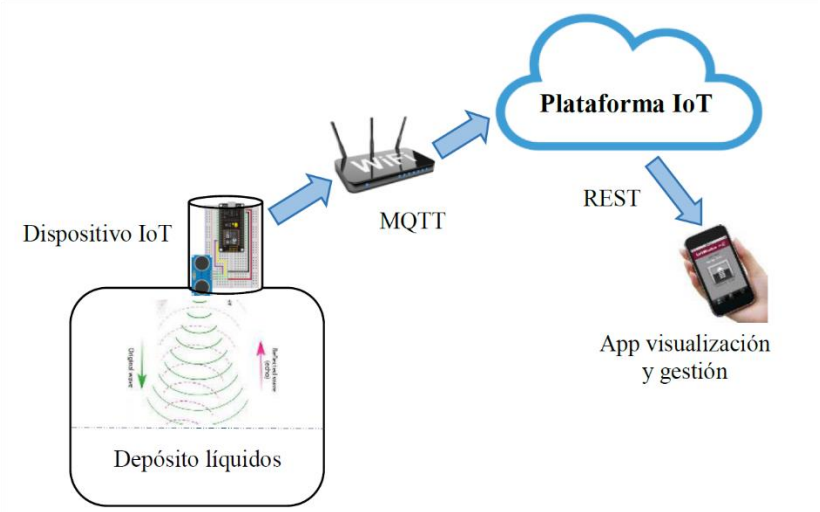


Figura 32: Esquema de funcionamiento de nuestra aplicación IoT

Concretamente los componentes hardware que se han utilizado son los siguientes:

- Placa de desarrollo **NodeMCU** con SoC **ESP8266**.

Como cualquier otra placa de desarrollo disponible en el mercado, NodeMCU tiene como principal objetivo facilitar el desarrollo de prototipos utilizando el microcontrolador que incorpora, facilitando el acceso a sus pines y su programación, ademas de aportar diferentes ventajas que dependen del propio kit de desarrollo.

La principal ventaja de NodeMCU es que incorpora un módulo WiFi que permite conectar nuestra placa a internet sin necesidad de ningun tipo de hardware adicional, por esto es una placa de desarrollo ideal para proyectos de Internet de las Cosas, ya que el 100% de estos proyectos van a requerir tener nuestro dispositivo conectado constantemente.

Como características de este kit de desarrollo podemos destacar las siguientes:

- Integra el módulo ESP12, basado en el SoC ESP8266, que a su vez monta el MCU Tensilica L106 de 32-bits (descrito en el capítulo II).

- Conversor Serie-USB: permite tanto alimentar la placa como programarla a través de un puerto USB (fundamental para no necesitar componentes adicionales).
- Memoria Flash de 4 MB.
- Fácil acceso a los pines del MCU.
- Pines de alimentación de 3.3V y 5V para alimentar otros componentes como sensores, leds, motores, etc.
- Dos LEDs de estado: uno para el módulo ESP12 y otro en la propia NodeMCU.
- Botón de “Reset” y “Flash”.

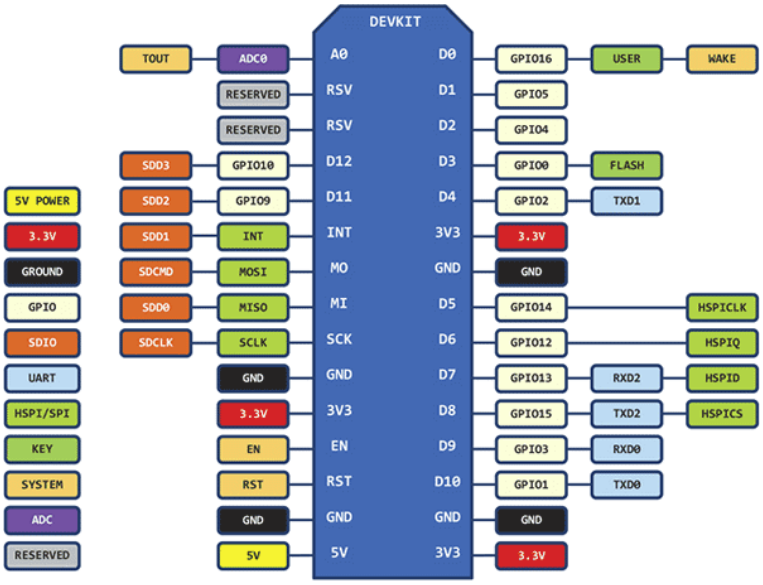


Figura 33: Pines de NodeMCU

La mejor forma de trabajar con este kit de desarrollo es utilizar una placa de prototipado. Asi podemos colocar nuestra placa y conectar facilmente los cables a los pines que deseemos. Como se puede ver en la figura 34, hay una gran cantidad de pines disponibles, aunque los más utilizados son:

- Los 13 pines digitales (D0 a D12).
- El pin analógico (A0).
- Los 3 pines de 3.3 V.
- El pin de 5V.
- Los 4 pines de tierra (GND)

Para comprobar su ubicación en la placa solo debemos buscar su posición teniendo como referencia la figura 33, aunque también está indicado en la propia placa la abreviatura de cada uno de los pines.

Es importante tener en cuenta que, debido a la carencia del SoC ESP8266 de memoria flash es imprescindible que NodeMCU utilice algunos de los pines de dicho chip para conectarlo a una memoria externa a dicho chip. Por este motivo hay algunos pines del NodeMCU que se recomienda no utilizar ya que, dependiendo del modelo y el fabricante, podrian estar reservados para esta finalidad y podria dar problemas si los utilizamos para otros propósitos. En concreto son los pines GPIO9 (D11 en NodeMCU) y GPIO10 (D12 en NodeMCU), por lo que lo más recomendable es tener esto en cuenta y evitar usarlos.

Por tanto, los principales motivos que nos han llevado a escoger NodeMCU como plataforma hardware para nuestro proyecto son, por un lado que nos soluciona directamente la carga de los programas y cómo conectar el otro elemento hardware que mencionamos a continuación directamente a nuestro MCU para su control, y por otro lado su capacidad de poder conectarse a redes WiFi. Con esto, sólo necesitaremos de un cable USB-microUSB y un ordenador con un entorno de desarrollo para programarla.

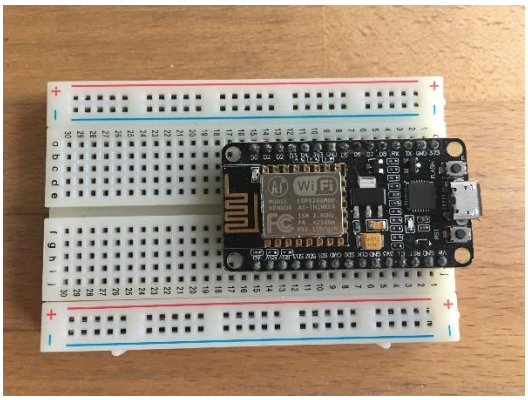


Figura 34: NodeMCU insertado en una placa de pruebas

- Sensor de ultrasonidos **HC-SR04**.

Este dispositivo nos permite medir distancias midiendo el tiempo entre pulsos de alta frecuencia (no audible por el ser humano) que rebotan en los objetos cercanos y son recogidos por el propio microfono que incorpora el sensor. Conociendo el tiempo que ha tardado el pulso emitido y mediante un sencillo cálculo podemos saber a qué distancia se encuentran los objetos que interaccionan con el sensor.

Como contrapartida es obligatorio mencionar que los sensores ultrasonidos son de baja precisión. La orientación de la superficie sobre la que rebota el pulso influye en la medición ya que puede ocasionar que la onda se refleje, falseando el resultado. Además no es aconsejable para casos en los que contemos con un gran numero de objetos porque pueden ocasionarse ecos y malas mediciones. Tampoco son adecuados para su uso al aire libre. Sin embargo para el propósito de nuestro proyecto es ideal, ya que es un entorno cerrado y la superficie del líquido siempre va a ser lisa y perpendicular a la dirección del pulso emitido.

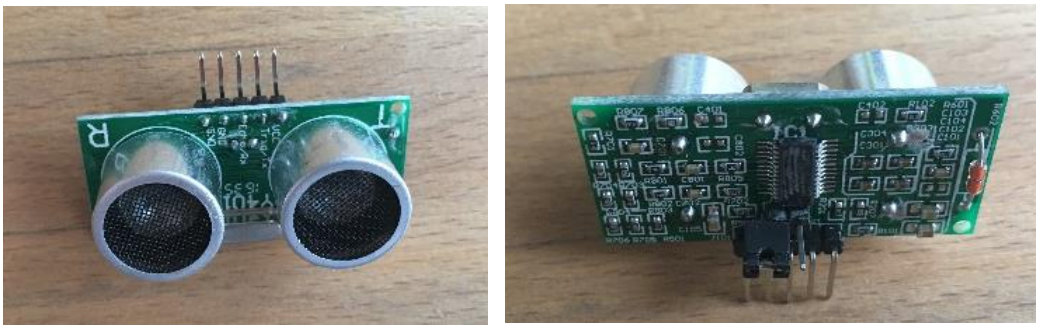


Figura 35: Sensor HC-SR04

Todo producto comercial en el ámbito del IoT tiene también una parte fundamental relacionada con el diseño de la carcasa que contendrá la electrónica en función de los requerimientos de la aplicación concreta donde se vaya a utilizar. Este aspecto también se ha de tener en cuenta en el prototipo inicial, y para ello se suelen diseñar carcasas personalizadas mediante **técnicas de impresión 3D**. De esta forma añadimos una nueva funcionalidad a nuestro prototipo, diseñando una carcasa que pueda incorporarse fácilmente a las bocas de los depósitos estándares que se pueden encontrar en nuestros hogares para el tipo de aplicación que se pretende desarrollar.

Para poder explotar nuestro dispositivo vamos a aprovechar las ventajas que nos ofrece una **Plataforma IoT**, lo que nos facilitará las tareas de transmisión, procesamiento, análisis y visualización de distintas estadísticas sobre los datos obtenidos por nuestro dispositivo conectado. De este modo lo único que habrá que realizar a nivel de software por parte del desarrollador será utilizar las herramientas que nos proporcione la Plataforma IoT, adaptándolas debidamente a nuestro hardware.

Para esto, **teniendo en cuenta los análisis que hemos realizado en el capítulo 3** de las distintas Plataformas IoT existentes y las ventajas que puede ofrecernos debido a la naturaleza y las necesidades de nuestro proyecto, emplearemos la Plataforma IoT **ThingSpeak**. Como ya hemos visto, es una Plataforma IoT muy recomendable para principiantes gracias a su intuitiva interfaz, pero que sin embargo posee potentes herramientas para satisfacer las necesidades de cualquier proyecto relacionado con el Internet de las Cosas a un nivel de tráfico de datos y dispositivos no muy exigente. Además si estuvieramos hablando de una aplicación con fines comerciales esta Plataforma nos permitiría ampliar nuestras funcionalidades si decidieramos abarcar un nivel de producción, con lo cual es de las Plataformas IoT más interesantes para cualquier finalidad.

Como alternativa a esta opción, en nuestro proyecto también vamos a experimentar con la otra alternativa que tendríamos a la hora de desarrollar un producto IoT, es decir, **crear nuestra propia aplicación** para acceder a nuestro dispositivo conectado. En este caso optaremos por crear una sencilla aplicación para dispositivos móviles que funcionen con el sistema operativo **Android**, a través de la cual podamos interactuar con nuestro dispositivo hardware de una manera parecida al método anterior mediante ThingSpeak.

4.2. Análisis de requisitos

En este subcapítulo vamos a analizar cuales serán los requisitos que tendrá nuestro prototipo. Los requistos son las propiedades o restricciones definidas que un proyecto debe satisfacer. Es una manera de establecer qué debe ser capaz de hacer nuestro prototipo.

4.2.1. Requisitos funcionales

Los requisitos funcionales son las utilidades que el prototipo va a proporcionar al usuario. En el caso de nuestro proyecto estos son:

- Obtener en tiempo real el porcentaje de contenedor que está ocupado por el fluido correspondiente (agua, gasoil, etc).
- Obtener un gráfico donde se represente cómo ha evolucionado el valor mencionado anteriormente a lo largo de un periodo de tiempo determinado.
- Realizar y mostrar visualmente estadísticas con los datos almacenados.
- Generar una alerta cuando el porcentaje disminuye de un determinado límite.
- Programar tareas de monitorización que permita obtener y almacenar el porcentaje anteriormente mencionado cada cierto periodo de tiempo (minutos, horas, días, etc).
- Visualizar gráficamente la localización del dispositivo.

4.2.2. Requisitos no funcionales

Los requisitos no funcionales describen aspectos del proyecto que están relacionados con el grado de cumplimiento de los requisitos funcionales:

- Cuando solicitamos el valor del porcentaje el tiempo de respuesta debe ser lo más pequeño posible.
- La visualización de los datos almacenados no debe dar lugar a ambiguedades.

- La alerta que nuestro dispositivo genere debe poder ser visible en un dispositivo móvil.
- El tamaño del kit de desarrollo hardware debe ser pequeño para permitir su integración en los depósitos.
- Asi mismo el diseño del producto final debe permitir la integración de manera sencilla en los depósitos.
- La aplicación de gestión y visualización debe poder comunicarse con el dispositivo desde cualquier lugar a través de Internet.
- La aplicación de gestión y visualización debe poder ser facilmente accesible por el usuario.

Capítulo V

Diseño del dispositivo IoT

En este apartado vamos a describir cómo vamos a construir nuestra plataforma hardware para nuestro proyecto a nivel de electrónica y componentes que vamos a emplear, además de explicar cómo hemos diseñado y construido la carcasa física que servirá de soporte para nuestro hardware.

5.1. Diseño del hardware del dispositivo IoT

Como hemos mencionado anteriormente vamos a utilizar el kit de desarrollo **NodeMCU** que lleva incorporado el MCU ESP8266. Además de las ventajas que ya hemos mencionado anteriormente sobre esta placa de desarrollo (ver sección 2.2.3), NodeMCU implementa la electrónica necesaria que nos permitirá controlar el microcontrolador del SoC ESP8266 de una manera más sencilla facilitando así el trabajo para crear nuestros prototipos.

Gracias a las características que nos proporciona NodeMCU podemos conectar directamente nuestra placa de desarrollo a nuestro PC para programarlo. Además esto nos permitirá alimentar a nuestro microcontrolador en la fase de despliegue con una fuente de 5V, mucho mas estandarizado que la alimentación de 3.3V que requiere un módulo ESP8266 sin kit de desarrollo.

El otro componente hardware indispensable para nuestro proyecto es el sensor ultrasónico **HC-SR04**, un dispositivo que emite ondas mecánicas cuya frecuencia

está por encima de la capacidad auditiva de los seres humanos (aproximadamente 20.000 Hz), que conectaremos con nuestra placa de desarrollo y configuraremos para obtener la distancia precisa al fluido en tiempo real, para posteriormente realizar la conversión y obtener el valor del volúmen que alberga el depósito.

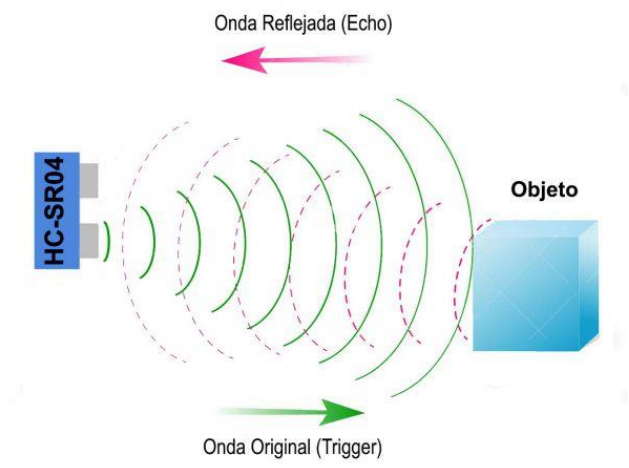


Figura 36: Esquema de funcionamiento del sensor HC-SR04

El funcionamiento del mismo es muy sencillo: generamos una onda sónica desde el sensor mediante un pulso en la patilla “Trig” (del inglés trigger o disparador). Dicha onda al encontrarse con un obstáculo rebotará en dirección opuesta a la inicial, volviendo así al sensor y quedando registrada mediante un pulso en la patilla “Echo”. De esta manera si contamos el tiempo que tarda entre la emisión de la onda original y la recepción de la onda reflectada y conociendo la velocidad del sonido podemos determinar la distancia a la que se encuentra el objeto con el cual ha interactuado la onda mediante la siguiente ecuación:

$$\text{Velocidad} = \text{Espacio}/\text{Tiempo} \rightarrow \text{Espacio} = \text{Velocidad} * \text{Tiempo}$$

Donde:

- **Velocidad** en nuestro caso es la velocidad del sonido, que es conocida y constante: 343m/s → 0.0343 cm/µs.
- **Espacio** es la incognita que queremos averiguar, la distancia (en centímetros) desde nuestro sensor al fluido que contiene el depósito.

- **Tiempo** es el periodo que tarda la onda (en microsegundos) desde que es originada por el disparador hasta cuando es recogida por el sensor nuevamente, es decir, el valor que obtendremos del HC-SR04.

Es muy importante tener en cuenta que la onda ha recorrido el camino dos veces (ida y vuelta) por lo tanto hemos de dividir el cálculo entre dos para conocer la distancia a la cual se encuentra el fluido. Por tanto la fórmula que emplearemos para obtener la distancia es la siguiente:

$$Espacio = (0.0343 * Tiempo)/2$$

$$Espacio = 0.01715 * Tiempo$$

Conociendo este dato únicamente tenemos que conocer las medidas del depósito sobre el que nuestro dispositivo esté trabajando para conocer el volúmen ocupado por el fluido que contenga. Estos datos dependerán del caso de uso concreto por lo tanto serán añadidos en la lógica de la Plataforma IoT que empleemos o en nuestra propia aplicación Android.

Una vez aclarados todos los aspectos físicos del problema vamos a describir cómo realizamos la conexión de los componentes mediante el uso de una placa de prototipado:

El montaje en sí es muy sencillo, tan solo debemos de prestar atención de alimentar correctamente nuestro sensor, para ello:

- Mediante el cable **rojo** conectaremos el pin Vcc (entrada del suministro de energía) a la salida +5V del NodeMCU.
- Mediante el cable **azul** los pines de tierra (GND) entre la placa de desarrollo y el sensor.

Una vez conectado el sensor correctamente para su alimentación tan sólo debemos de asignar los pines "*Trig*" y "*Echo*" del sensor a dos pines GPIO del NodeMCU, en nuestro caso hemos optado por el GPIO04 (D2 en NodeMCU) para el pin "Echo" del sensor (cable **amarillo**) y el GPIO05 (D1 en NodeMCU) para el pin "Trig" (cable **verde**) ya que estos pines no estan reservados para ninguna función adicional de la placa de desarrollo, con lo cual al no tener

problemas de falta de huecos libres porque no requerimos de una gran cantidad de conexiones en nuestro proyecto es más seguro utilizar estos pines libres.

Lo único que tendremos que tener en cuenta a nivel de software es configurar correctamente el pin GPIO04 como entrada, ya que es el que recibirá la comprobación de que la onda reflejada ha vuelto al sensor, y el pin GPIO05 como salida ya que es el encargado de emitir la señal.

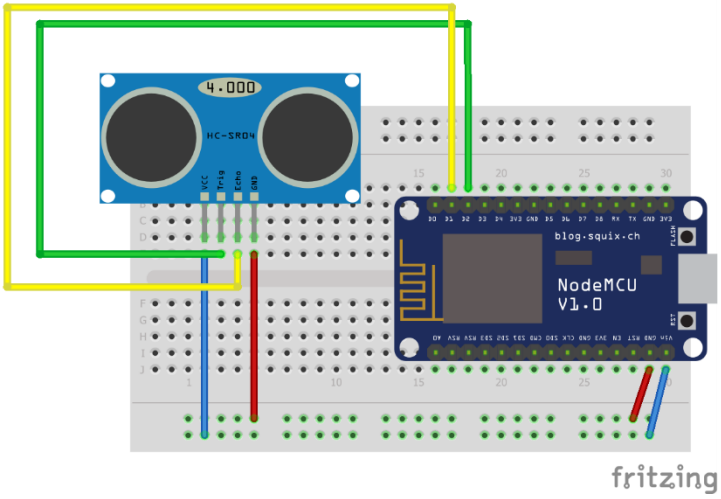


Figura 37: Esquema de conexión NodeMCU y HC-SR04

Una vez conectados correctamente todos los componentes del proyecto bastará con conectar mediante el puerto microUSB del NodeMCU a nuestro PC para cargar el programa. Este proceso se explicará más detalladamente en el próximo capítulo.

5.2. Diseño físico del producto

Para mejorar la apariencia y ofrecer una nueva funcionalidad a nuestro producto IoT hemos diseñado y construido una carcasa que servirá de soporte para nuestro prototipo. Como hemos dicho, aparte de mejorar estéticamente nuestro producto esta carcasa nos va a permitir instalarlo facilmente en los depósitos estándares que encontramos en los hogares actualmente. Para este fin hemos utilizado el software DesignSpark Mechanical.

Figura 38: Logo de DesignSpark Mechanical

DesignSpark es una herramienta gratuita de modelado 3D que permite diseñar prototipos y generar los archivos necesarios para imprimir mediante una impresora 3D el modelo diseñado. Es una herramienta sencilla que no requiere unos conocimientos altos en diseño asistido por computador (CAD). Por tanto permite que obtengamos modelos de una calidad excelente de manera fácil y rápida.

Sus principales características son:

- Curva de aprendizaje muy corta, conseguiremos obtener modelos de buena calidad en poco tiempo.
- Permite importar tanto ficheros de diseño en PCB como ficheros de diseño mecánico en 2D.
- Gran cantidad de librerias con más de 100 millones de modelos predefinidos que nos ayudarán a diseñar nuestros propios modelos.
- Permite exportar nuestros diseños en formato STL para impresión en 3D.
- DesignSpark es totalmente gratis de utilizar. Además la propiedad intelectual de cada uno de los proyectos que realicemos también nos pertenecerá.

Gracias a este software vamos a diseñar la carcasa de nuestro producto en dos piezas:

- Una **caja**, que contendrá nuestro kit de desarrollo NodeMCU y una abertura para permitir su alimentación a través del puerto microUSB del mismo.
- Una **base**, que permitirá acoplar nuestro dispositivo en el depósito a monitorizar mediante la rosca que trae el tapón de fábrica. Deberá contar con la abertura correspondiente para dejar salir la parte del sensor que

permite tanto generar la onda como el microfono para recogerla posteriormente.

En las siguientes figuras podemos ver el diseño de cada una de las partes por separado, además de sus respectivas dimensiones:

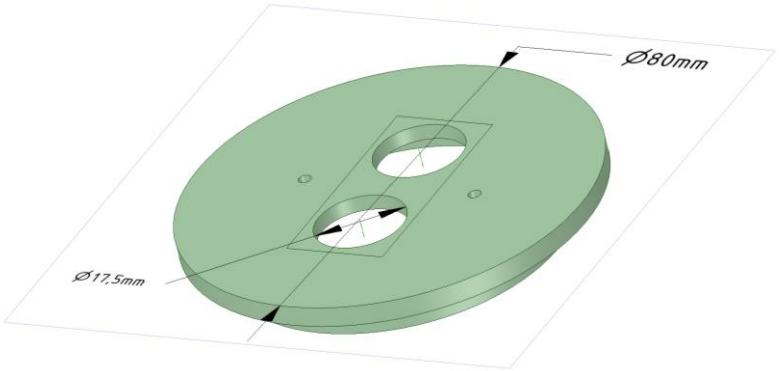


Figura 39: Pieza correspondiente a la base de nuestra carcasa

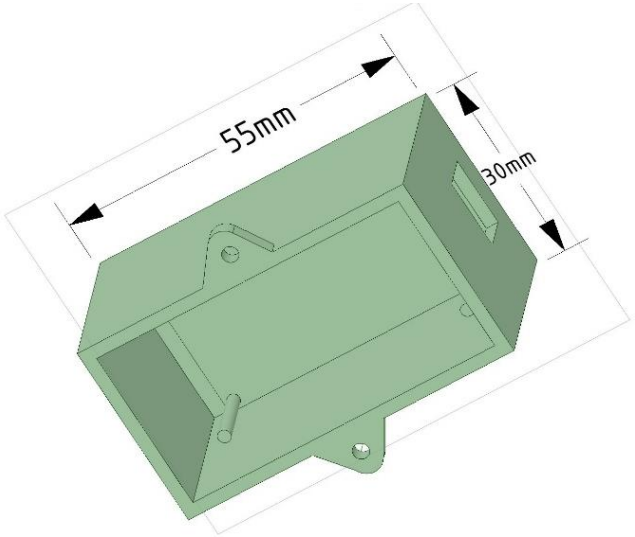


Figura 40: Pieza correspondiente a la caja de nuestra carcasa

Con este diseño hemos generado los archivos del modelo en formato STL y hemos realizado la impresión de cada uno de los objetos por separado. Por último hemos atornillado ambas piezas con el dispositivo NodeMCU y el sensor HC-SR04 en su interior, quedando de la siguiente manera:

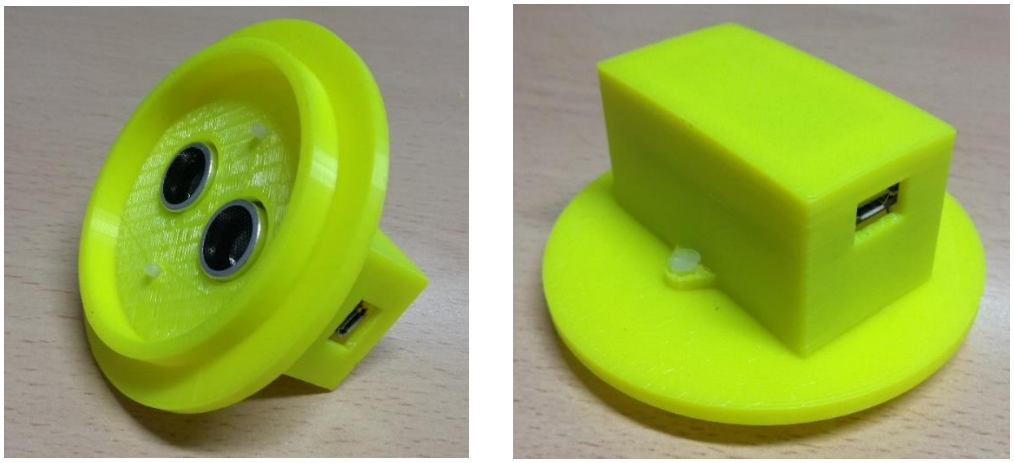


Figura 41: Aspecto final de nuestro dispositivo IoT

Además hemos probado a encajarlo en un depósito tal y como hemos descrito anteriormente y como se puede comprobar el resultado es satisfactorio:

Figura 42: Dispositivo anclado a un depósito real

Capítulo VI

Implementación del dispositivo IoT

Llega la hora de implementar la lógica necesaria que nos permita utilizar nuestro dispositivo NodeMCU para nuestro propósito. Este capítulo está dedicado a la programación del **software que se cargará en nuestro kit de desarrollo y realizará las tareas necesarias para llevar a cabo tanto la monitorización del depósito como la comunicación con la correspondiente herramienta de gestión y visualización**. La comunicación es el aspecto más importante que debemos resolver. Como explicaremos en el apartado correspondiente del capítulo, hemos implementado dos soluciones distintas que son completamente válidas para el objetivo de nuestro proyecto y nos permiten abarcar un mayor espectro de posibilidades para desarrollar la comunicación de este tipo de productos IoT. Lo veremos al final del capítulo.

El objetivo de nuestro proyecto era diseñar y construir un prototipo de producto IoT **completo y funcional** para monitorizar cualquier tipo de depósito que contenga cualquier tipo de líquido, ya sea agua, gasoil, aceite, etc. Hasta este punto ya tenemos el hardware necesario para llevar a cabo esta tarea, pero obviamente necesitamos programar nuestro dispositivo para conseguir nuestro objetivo.

En este capítulo vamos a mostrar **cómo podemos programar el microprocesador ESP8266** facilmente gracias a las ventajas (que mencionamos en el capítulo anterior) de la placa de desarrollo que hemos

escogido. Explicaremos el entorno de desarrollo que hemos utilizado para programarlo y también enseñaremos paso a paso cómo configurarlo correctamente para su uso con este tipo de placa de desarrollo. Además se explicará todo lo relativo al código que ejecutará nuestro dispositivo conectado.

6.1. Entorno de desarrollo

Como ya sabemos de capítulos anteriores vamos a emplear como dispositivo hardware la placa NodeMCU, que esta basada en el ESP8266. Como sabemos, una de las ventajas de esta placa de desarrollo es la compatibilidad con el **entorno de desarrollo de Arduino**, gracias a lo cual podemos conectar NodeMCU a nuestro ordenador y trabajar con él desde dicho entorno como si se tratase de cualquier placa de la marca Arduino.

Sin embargo, para poder utilizar el IDE de Arduino con nuestra placa NodeMCU debemos realizar una **configuración previa** que vamos a explicar paso por paso en el Anexo C.

El funcionamiento de este entorno de desarrollo es muy sencillo. Lo mas importante que debemos conocer son las distintas partes o secciones que lo forman:

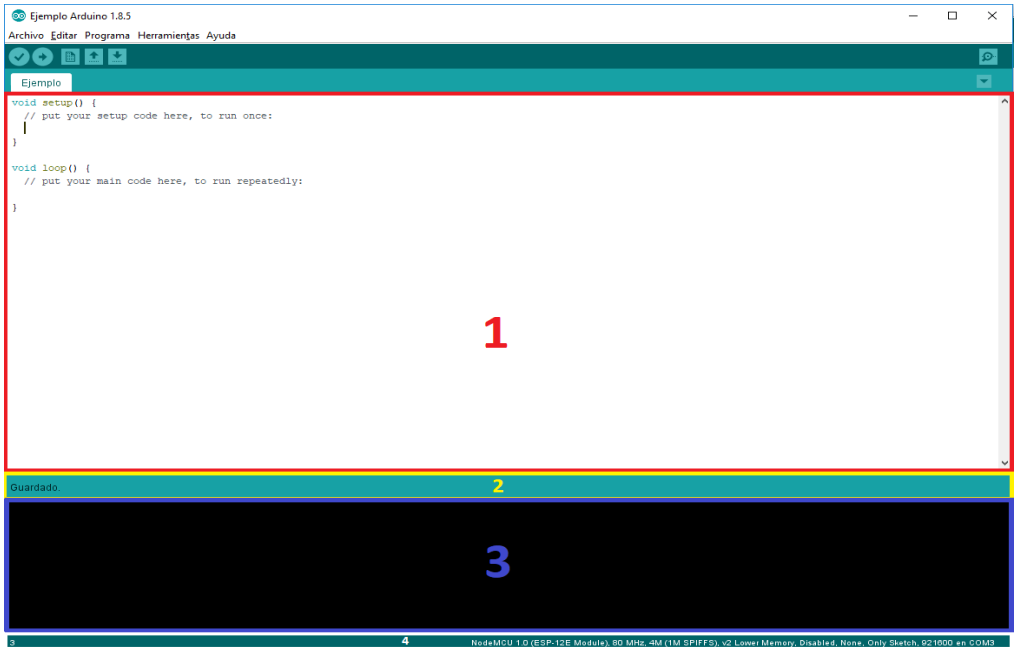


Figura 43: Esquema de las partes del IDE Arduino

Comenzamos con la parte del **editor de código** (marcado con un 1 en la figura 43). Aquí es donde vamos a fijar nuestra atención la mayor parte del tiempo, pues es donde escribiremos el código de nuestro programa. En la parte de arriba del editor encontramos el nombre del archivo (en este caso el archivo se denomina “Ejemplo”) y más arriba encontramos el menú de acceso rápido donde encontramos las funciones principales del programa:

 Compilar: permite, primero guardar el código en el disco duro de nuestro ordenador, después verificar el código para detectar posibles errores y, si no se encuentra ningun error, compilar el programa. Preparandolo así para cargarlo en nuestra placa de desarrollo en lenguaje máquina.

 Subir: como su propio nombre indica, nos permite subir o cargar el programa compilado a nuestra placa hardware conectada a través del puerto USB a nuestro ordenador. Si no se ha compilado antes primero lo compila automaticamente si se ha efectuado algún cambio en el código.

 Nuevo: genera una nueva ventana con un nuevo programa por defecto (sólo con las funciones setup y loop vacías).

 Abrir: abre un programa que ya exista y se encuentre guardado en nuestro ordenador.

 Salvar: guarda el código actual en el disco duro de nuestro ordenador, sin compilar el programa.

 Monitor serie: muestra los mensajes de comunicación entre el ordenador y la placa de desarrollo durante la ejecución del programa cargado.

Todas estas opciones las podemos encontrar también (aunque un poco más ocultas) dentro de los respectivos menús de la barra de herramientas, y también con sus respectivos atajos de teclado.

Por debajo del editor encontramos otra parte importante que podriamos denominar como el **área de notificaciones** (marcado en la figura 43 con el número 2), donde se nos mostrará la última acción realizada o la acción que se está ejecutando actualmente (“Guardado”, “Compilado”, “Subido”, “Subiendo…”, etc).

Debajo de las dos partes anteriores encontramos la **consola** o **terminal** (representado en la figura 43 con un 3), que nos ofrecerá información relativa al control del IDE sobre el programa que estamos desarrollando. Entre otras cosas nos puede mostrar los errores a la hora de compilar el programa, cómo evoluciona el proceso de carga sobre la placa o si se produce algún fallo de comunicación entre el ordenador y la placa de desarrollo.

Por último, la parte más inferior de la interfaz del IDE nos muestra una barra de estado (podemos identificarla marcada con el número 4 en la figura 43) donde encontramos dos datos interesantes. Por un lado en la parte izquierda se indica el número de fila donde tenemos situado el cursor dentro del código que hay en el editor. Por otro lado y más importante es lo que se muestra en la zona derecha de esta barra, el nombre de la placa de desarrollo que tenemos actualmente conectada (es decir, el dispositivo hardware al cual vamos a cargar el programa que estemos escribiendo en ese momento) y el puerto al cual se encuentra conectada dicha placa. Esto nos permite saber rápidamente con qué placa nos encontramos trabajando, en caso de que lo estuvieramos haciendo con más de una a la vez.

El lenguaje de programación de alto nivel que hay que utilizar para poder utilizar el IDE de Arduino es **C++**, ya que es el lenguaje que comprende el compilador para traducir a lenguaje máquina, que serán las instrucciones que se cargarán en el MCU de nuestra placa de desarrollo. Para ser exactos se trata de un subconjunto de las funciones de C/C++ estandar adaptadas y con algunas otras funciones nuevas, pero al final se utiliza un compilador de lenguaje C por lo que todas las funciones soportadas por el compilador podrán ser compiladas con el IDE de Arduino. En concreto, el compilador que utiliza el IDE se denomina **AVR-GCC** y, como hemos dicho, su función es traducir el lenguaje de alto nivel que utilizamos para escribir sentencias de código en el editor al lenguaje máquina

hexadecimal que interpretará nuestra placa, en nuestro caso NodeMCU. Para poder cargar estas instrucciones en la memoria flash de la placa el IDE utiliza un software que se instala junto al mismo entorno de desarrollo denominado **AVRdude**, que junto con el software **Bootloader** que se encuentra por defecto cargado en el circuito integrado de nuestra placa se encargarán de realizar este proceso automaticamente. En resumen, nosotros solo debemos preocuparnos de escribir en el editor utilizando como lenguaje C++ el código que queremos ejecutar en nuestro dispositivo hardware y utilizar la función de cargar el programa que nos facilita el entorno de desarrollo.

Para cualquier duda sobre las funciones y librerias disponibles en el entorno de desarrollo es recomendable visitar la **Referencia oficial de Arduino** (Arduino, s.f.), donde encontraremos toda la información necesaria sobre el lenguaje, funciones, variables y estructuras soportadas.

El entorno de desarrollo nos proporciona las ventajas tipicas de este tipo de software: nos facilita la escritura del código destacando las palabras reservadas, permite contraer estructuras y funciones, la gestión de librerias y formatear nuestro código. Pero lo más importante es que nos permite compilar el código y cargar nuestro programa (también denominado “sketch”) a nuestra placa hardware. Los programas creados con el IDE de Arduino se guardan con el nombre de dicho programa y la extensión “.ino” dentro de una carpeta con el mismo nombre que el programa. Esto lo hace automaticamente el entorno de desarrollo cuando compilamos por primera vez un nuevo programa, dandonos a elegir la ubicación del archivo dentro de nuestro ordenador.

El esquema que siguen todos los programas que creamos con el IDE de Arduino es muy sencillo. Todos constan de dos partes bien diferenciadas:

- Función **Setup()**: Es la función que se ejecuta en primer lugar cuando arranca nuestro programa. Solo se ejecuta una vez y es el lugar indicado para cargar datos, iniciar variables y realizar cualquier ajuste inicial que requiera nuestro programa que sólo necesiten ser ejecutadas una sola vez.

- Función **Loop()**: Es la función principal de nuestro programa y se ejecuta directamente después de la función Setup, pero a diferencia de esta, la función Loop se ejecuta constantemente de forma cíclica. Es decir cuando se termina de ejecutar todas las sentencias de esta función se vuelve al principio de ella entrando asi en un bucle infinito. Aquí es donde escribiremos el grueso del código que dotaran de inteligencia a nuestro dispositivo hardware.

```
1.  void setup() {
2.    // put your setup code here, to run once:
3.
4.  }
5.
6.  void loop() {
7.    // put your main code here, to run repeatedly:
8.
9.  }
```

Por supuesto, además de estas también podemos crear nuestras propias funciones y hacer referencia a ellas dentro de las dos anteriores, pero estas son fundamentales y deben estar en nuestros programas para un correcto funcionamiento. Además también es posible declarar variables o estructuras fuera de la función setup, pero es aconsejable inicializarlas dentro de dicha función.

También podemos importar librerias que nos proporcionarán funcionalidades extras para utilizar en nuestros programas. Por defecto el IDE se instala junto a un buen número de librerias, pero también podemos descargarlas mediante la herramienta de Gestión de librerías del propio entorno de desarrollo (dentro de "**Programa > Incluir librería**") o directamente desde internet. Para instalarlas tendremos tres métodos diferentes: mediante el gestor (lo más aconsejable), a partir de un archivo zip o manualmente incluyendo los archivos nosotros mismos en la ubicación correspondiente. Para más información sobre el uso de librerías en el IDE de Arduino visitar su web oficial donde se explica detalladamente www.arduino.cc/en/Guide/Libraries

6.2. Código de NodeMCU

Una vez que tenemos claro cómo funciona el entorno de desarrollo que vamos a utilizar procederemos a desarrollar el código que ejecutará nuestro dispositivo conectado, de manera que ejecute las tareas necesarias para la monitorización del depósito y podamos comunicarnos con la aplicación de gestión y visualización.

Como ya hemos visto en la sección anterior, para escribir el código que se ejecutará en nuestro kit de desarrollo *NodeMCU* utilizamos el entorno de desarrollo de Arduino configurado correctamente para trabajar con nuestro prototipo. Teniendo esto en cuenta podemos empezar a escribir el código de nuestro programa.

El primer paso será implementar la lógica necesaria para **manejar el sensor** *HC-SR04* que conectamos a nuestro kit de desarrollo. Para ello debemos emplear dos pines específicamente para el control del sensor:

- **TriggerPin**: es el pin conectado a la patilla “Trig” del sensor, cuya función será activar el pulso que generará la onda que posteriormente recogeremos para conocer la distancia (ver figura 36 en el capítulo 5.1). Para ello tendremos que declararlo como pin de salida (Output). En nuestro caso hemos dedicado esta función al pin D1 que coincide con el GPIO05 en el SoC ESP8266.

- **EchoPin**: el pin conectado a la patilla “Echo” del sensor. Su función es escuchar hasta que la onda emitida mediante el *Trigger* o disparador rebote en el obstaculo y vuelva al sensor, midiendo el tiempo transcurrido. Para esto debemos declarar este pin como de entrada (*Input*). En nuestro caso hemos utilizado para éste menester el pin D2 que coincide con el GPIO04 en el SoC ESP8266.

Por tanto tendremos primero que declarar como constantes los pines que vamos a utilizar y después establecerlos como pines de salida o entrada mediante la función *pinMode* cuyo funcionamiento es muy intuitivo como podemos ver:

```
1.  const int EchoPin = 4; // GPIO04 en ESP8266 | D2 en NodeMCU
2.  const int TriggerPin = 5; // GPIO05 en ESP8266 | D1 en NodeMCU
3.
4.  void setup() {
5.
6.    pinMode(TriggerPin, OUTPUT);
7.    pinMode(EchoPin, INPUT);
8.
9.  }
```

Ahora que tenemos bien configurados los pines que van a comunicarse con nuestro sensor podemos implementar las acciones para obtener la distancia existente entre nuestro dispositivo y el obstáculo que encuentre (en nuestro caso será el líquido contenido en el depósito, o en su defecto el fondo del depósito), para esto nosotros hemos implementado el método *getDistance*:

```
1.  float getDistance(int TriggerPin, int EchoPin){
2.    float length, distance;
3.
4.    // Ponemos a LOW durante 4 microsegundos para generar un pulso limpio
5.    digitalWrite(TriggerPin, LOW);
6.    delayMicroseconds(4);
7.
8.    // Generamos onda
9.    digitalWrite(TriggerPin, HIGH);
10.   delayMicroseconds(10);
11.
12.   digitalWrite(TriggerPin, LOW);
13.
14.   // Medimos el tiempo entre pulsos, en microsegundos
15.   length = pulseIn(EchoPin, HIGH);
16.
17.   // Convertimos a distancia, en cm
18.   distance= length * 10 / 292 / 2;
19.
20.   return distance;
21. }
```

Como vemos el método es muy sencillo, simplemente debemos activar el pin *Trigger* y posteriormente activar el pin de *Echo* que nos devolverá el tiempo transcurrido, con lo cual con un simple cálculo (ver apartado 5.1) obtendremos la distancia entre el dispositivo y el obstáculo.

El siguiente punto que tenemos que implementar es **cómo realizar la conexión de nuestro dispositivo a la red WiFi**, algo indispensable para el desarrollo de nuestro proyecto. Por suerte este proceso es sencillo gracias a la clase WiFi de la librería *ESP8266WiFi* que debemos incluir en nuestro programa importanto el correspondiente archivo cabecera “ESP8266WiFi.h”. Para configurar nuestra conexión con la clase WiFi tendremos que especificar los siguientes parámetros:

- **SSID**: el identificador de la red WiFi a la cual nos conectaremos.
- **Password**: la contraseña de la red con el SSID especificado.

De la misma manera necesitamos tres parámetros adicionales para establecer una *IP* privada estática para nuestro dispositivo conectado, cosa que es muy importante para asegurar la comunicación estable entre la aplicación de gestión y el propio dispositivo:

- **IP**: dirección *IP* privada que queremos asignar a nuestro dispositivo (debemos asegurarnos que esta *IP* se encuentra libre en nuestro router).
- **Gateway (puerta de enlace)**: es la dirección privada dentro de nuestra red de nuestro router o puerta de enlace.
- **Subnet (mascara de red)**: es la mascara de red que sirve para delimitar el ámbito de nuestra red, normalmente utilizaremos una red de clase C cuya mascara corresponde con 255.255.255.0 estableciendo un número máximo de hosts de 254.

Con todos estos parámetros podemos configurar la conexión a la red mediante la red *WiFi* de nuestro router. Para ello además de declarar estas constantes debemos codificar lo siguiente en la función *setup*:

```
1.  const char* ssid = "miRed";
2.  const char* password = "miContraseña";
3.
4.  IPAddress ip(192, 168, 1, 104);
5.  IPAddress gateway(192, 168, 1, 1);
6.  IPAddress subnet(255, 255, 255, 0);
7.
8.  void setup() {
9.
10.   WiFi.mode(WIFI_STA);
11.   WiFi.disconnect();
12.
13.   WiFi.config(ip, gateway, subnet);
14.
15.   // Conectar a la red WiFi
16.   WiFi.begin(ssid, password);
17.
18.   while(WiFi.status() != WL_CONNECTED){
19.     delay(500);
20.   }
21.
22.   // Conexión establecida
23.   Serial.println("\nConectado");
24.
25. }
```

Como podemos observar los pasos a realizar para conectar con nuestra red *WiFi* son muy sencillos. En primer lugar debemos **establecer nuestro dispositivo en modo *Estación* (STA)**, para conectar con una red *WiFi* establecida por un punto de acceso (en nuestro caso el punto de acceso será nuestro router), mediante el método *mode* de la clase *WiFi*. Después debemos **añadir los parámetros que definen la dirección *IP* privada** que queremos asignar de forma estática a nuestro dispositivo, mediante el método *config*, y por último llamaremos al método *begin* para **establecer la conexión**, pasando como parámetros tanto el *SSID* como la contraseña de la red. Después de que la secuencia de ejecución del programa supere la estructura de control *while* ya se habrá establecido la conexión con nuestro punto de acceso a Internet.

Otro aspecto importante a tener en cuenta en el programa que ejecutará nuestro dispositivo conectado es cómo **almacenar los datos indispensables** para prevenir la desconexión del dispositivo de la fuente de alimentación. Para esto utilizaremos la memoria *Flash* de nuestro kit de desarrollo *NodeMCU*, mediante la clase *EEPROM* de la librería que estamos utilizando. Esta clase nos proporciona métodos mediante los cuales podemos acceder a una dirección de memoria concreta y almacenar u obtener datos a partir de esa posición, dependiendo del tipo de dato que queramos almacenar. Para esto antes de nada debemos añadir una instrucción en la función *setup* de nuestro programa para inicializar la clase indicando el tamaño de la memoria de nuestro dispositivo (en nuestro caso disponemos de 4 MB) mediante el método *begin* de la clase *EEPROM*. Una vez hecho esto podremos utilizar los métodos *get* y *put* para leer o escribir en la memoria. Para ilustrar esto podemos ver el siguiente ejemplo:

```
1.  int dato;
2.  String dato2;
3.
4.  void setup() {
5.
6.    EEPROM.begin(4096);
7.
8.    // Recuperar datos guardados en memoria EEPROM
9.    EEPROM.get(0, dato);
10.   EEPROM.get(sizeof(int), dato2);
11.
12. }
13.
14. void loop(){
15.
16.  dato = actualizarDato();
```

```
17.   dato2 = actualizarDato2();
18.
19.   // Escribimos datos en memoria EEPROM
20.   EEPROM.put(0, dato);
21.   EEPROM.put(sizeof(int), dato2);
22.
23.   // Guardamos los cambios realizados
24.   EEPROM.commit();
25.
26. }
```

Como se puede observar para utilizar los métodos *get* y *put* debemos indicar como primer parámetro la dirección de memoria a partir de la cual queremos leer/escribir, y como segundo parámetro la variable a la cual queremos asignar el dato guardado en la memoria o que queremos guardar en la misma. Además de los tipos básicos de variables podemos guardar y recuperar estructuras personalizadas de la misma forma.

Es importante tener claro cuales son los datos que queremos guardar y cómo vamos a organizarlos en la memoria *Flash* de nuestro dispositivo, ya que debemos seguir un orden a la hora de escribir y recuperar los datos almacenados. Nosotros vamos a seguir la siguiente estructura a la hora de organizar la memoria:

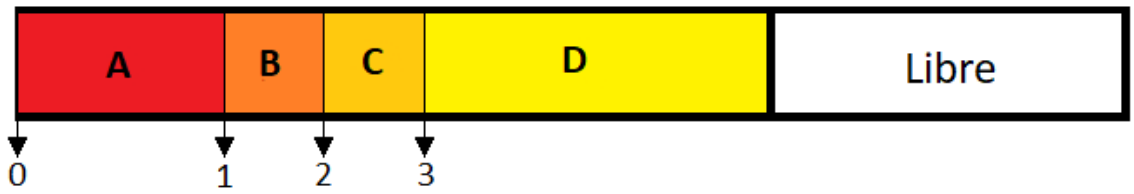


Figura 44: Estructura de los datos guardados en la memoria del dispositivo IoT

Los elementos que almacenamos en la memoria están representados en la figura anterior con letras y son los siguientes:

- **A: Estructura "contactPref"** que alberga tanto el email como el nivel mínimo (en porcentaje) del depósito.
- **B: Entero "n_days"**, que representa el número de días almacenados que se encuentran en espera de recibir la petición de actualización por parte de la aplicación de gestión y almacenamiento.
- **C: Entero "pointer"**, que es utilizado en la aplicación para mantener ordenado el array que almacena las distancias de cada día.

- **D: Array de tipo Float** donde almacenamos las distancias recogidas por el sensor cada día.

Por otra parte, las posiciones de memoria a la cual debemos acceder para obtener cada elemento se representan en la misma figura con número enteros:

- **0**: Es la posición para acceder al elemento A y se corresponde con la dirección de memoria 0.
- **1**: Es la posición para acceder al elemento B y se corresponde con la dirección de memoria:

$$sizeOf(contactPref)$$

- **2**: Es la posición para acceder al elemento C y se corresponde con la dirección de memoria:

$$sizeOf(contactPref) + sizeOf(int)$$

- **3**: Es la posición para acceder al elemento D y se corresponde con la dirección de memoria:

$$sizeOf(contactPref) + 2\left(sizeOf(int)\right) + (\boldsymbol{pointer} * sizeOf(float))$$

  Donde "pointer" es la posición que ocupa en el array el dato al cual queremos acceder.

Por último debemos establecer un modo de controlar cada cuanto tiempo se realizan las tareas necesarias en nuestro dispositivo IoT. Como hemos visto anteriormente el código implementado en el método *loop* de nuestro programa se ejecuta cíclicamente de manera indefinida, por tanto debemos establecer algún mecanismo para controlar la ejecución de las acciones que no requieren estar ejecutandose constantemente, concretamente:

- **Cada 24 horas** almacenar la distancia que nos servirá para averiguar el porcentaje de volumen del depósito que ocupa el líquido.

- **Cada 6 horas** comprobar si el nivel se encuentra por debajo del nivel mínimo establecido por el usuario, y en caso afirmativo enviar la debida notificación al correo electrónico indicado.

Existen varias formas de hacer esto, lo más habitual es utilizar un **RTC** (**Real Time Clock**) que mantenga la hora actualizada en cada momento, pero nosotros no contamos con este componente en nuestro kit de desarrollo por lo cual vamos a utilizar una solución distinta. Teniendo en cuenta el tiempo transcurrido desde que el dispositivo arrancó podemos establecer intervalos de tiempo para realizar las acciones que hemos mencionado previamente. Con la función *millis* obtenemos los milisegundos que han transcurrido desde el arranque. Este dato es almacenado internamente y se sobrepasa (es decir, vuelve a cero) cuando transcurren 50 días. Como no necesitamos establecer intervalos que superen este límite podemos proceder en cada iteración de *loop* de la siguiente manera:

```
1.   unsigned long interval = 86400; // 24 horas en segundos
2.   unsigned long interval2 = 21600; // 6 horas en segundos
3.   unsigned long previousSeconds = 0;
4.   unsigned long previousSecondsEmail = 0;
5.
6.   void loop() {
7.    unsigned long currentSeconds = millis() / 1000;
8.
9.    if(currentSeconds - previousSeconds >= interval){ // Ciclo de 24 horas
10.     previousSeconds = currentSeconds ;
11.
12.     // Añadir dato al array en memoria
13.   }
14.
15.   if(currentSeconds - previousSecondsEmail >= intervalEmail){ // Ciclo de 6 horas
16.     previousSecondsEmail = currentSeconds ;
17.
18.     // Comprobar nivel del depósito y comparar con el mínimo
19.   }
20.  }
```

Es muy importante tener en cuenta que, como el valor máximo de un tipo **unsigned long** es 4.294.967.295 (0xffffffff), tenemos que hacer los cálculos en segundos puesto que si asignamos a una variable de tipo unsigned long el valor de *millis* directamente llegará un punto en que el valor asignado será ligeramente mayor al límite del tipo unsigned long (recordamos que millis se desborda, es decir vuelve a cero, cuando llega a 50 días que se corresponde con 4.320.000.000 milisegundos) por lo cual se desbordaría el dato y podría producir errores. Por esta razon a la hora de inicializar el valor a la variable

currentSeconds le asignamos el valor de *millis* dividido entre 1000 (1 segundo = 1000 milisegundos).

6.3. Comunicación con la herramienta de gestión y visualización

Como veremos en el capítulo siguiente, vamos a implementar dos alternativas distintas para gestionar y acceder a los datos que recogerá nuestro dispositivo durante su ejecución. Para entender mejor este apartado vamos a adelantar que utilizaremos tanto una **aplicación para dispositivos móviles** con sistema operativo Android como una **Plataforma IoT**, aunque esto lo justificaremos mejor en el siguiente capítulo dedicado en su totalidad a dicha aplicación de gestión y visualización.

Para poder establecer la conexión con dicha herramienta o aplicación que vayamos a utilizar debemos codificar esta característica de diferentes formas en el programa de nuestro dispositivo NodeMCU. Por esta razón es imprescindible en este punto realizar una bifurcación en nuestra exposición para poder explicar cómo implementar ambas alternativas en el programa del dispositivo conectado de una manera más sencilla y comprensible de cara al lector.

6.3.1. Comunicación con una aplicación movil

Para conseguir comunicar nuestra propia herramienta de visualización y gestión desarrollada como aplicación para dispositivos móviles con nuestro dispositivo conectado que se encontrará instalado en el depósito que deseamos monitorizar vamos a utilizar un Servicio Web.

Un **Servicio Web** es una tecnología que mediante un conjunto de protocolos y estándares permite intercambiar datos entre aplicaciones que pueden ser muy diferentes entre sí, independientemente de las características particulares de cada aplicación como la plataforma o el lenguaje de programación.

El Consorcio World Wide Web, también conocido como **W3C** de su nombre en inglés **World Wide Web Consortium** define los Servicios Web como “una

aplicación de software identificada por un *URI* (Uniform Resource Identifier), y cuyas interfaces se pueden definir, describir y descubrir mediante documentos XML. Hacen posible la interacción entre aplicaciones software mediante mensajes intercambiados a través de Internet mediante las reglas que definen sus protocolos".

Las principales ventajas de los Servicios Web frente a otras tecnologías son:

- Emplean estándares de Internet como HTTP o XML.
- Permiten interoperabilidad entre aplicaciones de software muy distintas entre sí, independientemente de sus características individuales.
- Fomentan los estandares y protocolos basados en texto, que facilitan el acceso al contenido.
- Permiten combinar servicios de software ubicados en diferentes lugares para proporcionar servicios integrados.

Dos de los Servicios Web más populares son los llamados SOAP y REST. Los Servicios Web SOAP se basan en el intercambio de mensajes que siguen el estandar Simple Object Access Protocol (W3C, SOAP Specifications, 2018). Este formato de mensaje es muy utilizado por una gran variedad de aplicaciones y sistemas.

Por otro lado, el denominado Servicio Web REST (Representational State Transfer) se basa en el popular estándar HTTP, por lo cual permite crear servicios que puedan ser utilizados por cualquier aplicación o sistema que entienda este protocolo (W3C, REST Semantic Web Standards, 2001). Es una forma muy ligera y sencilla de implementar un Servicio Web. En un servicio REST para acceder a los recursos de la red se emplea un identificador URI (Uniform Resource Identifier, identificador de recursos uniforme), que no es más que una cadena de caracteres que apunta a un recurso de la red de forma inequivoca.

A los servicios REST se le pueden añadir características de encriptación, seguridad, gestión de sesiones, etc. ya que no podemos olvidar que funciona a

través de HTTP, por lo cual se pueden añadir funciones como establecer contraseñas, usuarios, SSL, etc.

Para manipular los recursos HTTP se proporcionan los siguientes métodos:

- **GET**: Para consultar y leer recursos.
- **POST**: Para crear recursos.
- **PUT**: Para editar recursos.
- **PATCH**: Para editar partes concretas de un recurso.
- **DELETE**: Para eliminar recursos.

Cuando se envia una petición HTTP se recibirá una respuesta que incluirá un código de estado. HTTP tiene un abanico de códigos que toda aplicación basada en dicho protocolo deberá utilizar. Dichos códigos cubren todas las posibilidades que pueden darse a la hora de incluir una respuesta según como se ha desarrollado la operación recibida por la petición. Los grupos de códigos de estado son los siguientes:

Código de estado	Significado
2xx	Peticiones **correctas**. Indica que la petición fue recibida, entendida y aceptada.
3xx	**Redirecciones**. El cliente debe tomar una acción adicional para completar la petición.
4xx	**Errores del cliente**. La solicitud contiene sintaxis incorrecta o no puede procesarse.
5xx	**Errores de servidor**. Indican que el servidor falló a la hora de completar una solicitud aparentemente correcta sintácticamente.

Por otro lado, el contenido de los mensajes suele estar en formato *XML*, aunque también es válido utilizar un formato *JSON*.

Como hemos mencionado anteriormente, para acceder a los recursos utilizaremos un identificador URI. La estructura usual que mantienen estos identificadores es de directorios, generando una jerarquía en los recursos. Además todo identificador debe cumplir una serie de requisitos para ser adecuado:

- Debe ser único, de modo que permita identificar cada recurso con únicamente un URI.
- Debe escribirse en minúsculas, con la excepción del identificador de método que vamos a emplear (GET, PUT, POST, etc).
- No introducir espacios en blanco. En su lugar utilizar preferentemente guiones o guiones bajos.
- Los identificadores para una serie de recursos deben mantener una jerarquía lógica.
- No deben implicar una acción. Por tanto no se debe utilizar verbos para formarlos.

Para nuestro proyecto hemos decidido utilizar este tipo de Servicio Web debido a su sencillez y las ventajas que hemos mencionado anteriormente. Para lograr la comunicación con el dispositivo debemos implementar las siguientes peticiones:

URI	Método HTTP	Descripción
/nivel_actual	GET	Petición para obtener el volumen actual que ocupa el líquido en nuestro depósito.
/registro	GET	Petición cuyo objetivo es en primer lugar conocer si existen datos guardados por parte del dispositivo conectado referentes a los niveles del depósito durante los últimos días, y en caso afirmativo se envían a la aplicación para su procesamiento y almacenamiento en la base de datos.

/preferencias	PUT	Petición que envía al dispositivo conectado los datos referentes a las preferencias de monitorización introducidas por el usuario

Para poder recibir y contestar peticiones *HTTP* debemos configurar *NodeMCU* como un **servidor web** capaz de recibir, procesar y contestar a este tipo de peticiones.

Para esto únicamente tendremos que crear una instancia de la clase *WiFiServer* estableciendo como puerto de escucha el puerto 80 e iniciar el servidor una vez que hayamos establecido conexión con nuestro router:

```
1.  // Creamos una instancia del servidor especificando el puerto de escucha
2.  WiFiServer server(80);
3.
4.  void setup() {
5.
6.   // Establecer conexion WiFi
7.
8.   // Empezar el servidor
9.   server.begin();
10.  Serial.println("Servidor empezado");
11. }
```

Ahora que tenemos nuestro dispositivo escuchando y listo para recibir peticiones debemos implementar la lógica que detecte cuando se reciba una petición. Esto se implementará en la función *loop* de nuestro programa, que como ya sabemos se ejecuta cíclicamente de manera ilimitada. El proceso a seguir es comprobar en cada ciclo de la función *loop* **si existe algún cliente transmitiendo una petición a nuestro dispositivo, en caso afirmativo comprobaremos de qué tipo de petición se trata y la procesaremos** de la forma adecuada a cada caso. En caso negativo simplemente terminamos la ejecución del ciclo actual de la función *loop*:

```
1.  void loop() {
2.   // comprobar si existe un cliente conectado
3.   WiFiClient client = server.available();
4.
5.   if(!client)
6.     return; // No hay petición, terminamos el ciclo
7.
8.   // Nueva petición recibida
9.  }
```

Como podemos ver mediante la función *available* de la clase *WiFiServer* obtenemos una **referencia a un cliente que está enviando una petición** a nuestro dispositivo (en caso de que exista algún cliente transmitiendo, ya que en caso contrario se le asignará un *null* al objeto client) y a partir de ese momento ya podemos proceder a su procesamiento.

El **procesamiento** consiste en obtener la petición en formato cadena de caracteres y conocer cual de los métodos disponibles para nuestro proyecto (los cuales hemos visto antes en esta misma sección) es el que tenemos que aplicar. Después solamente tendremos que obtener o almacenar en la memoria del dispositivo los datos, crear la respuesta apropiada y enviarla al cliente mediante el método print de la clase *WiFiClient*.

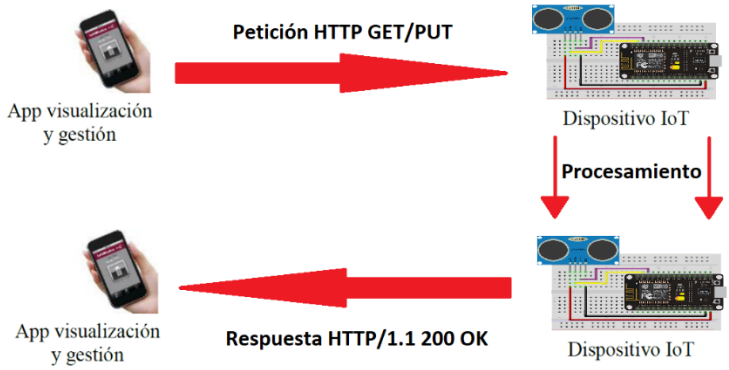


Figura 45: Esquema de comunicación entre la aplicación y el dispositivo IoT

Cabe mencionar que todos los cálculos matemáticos necesarios para obtener el volumen a partir de la distancia recibida del dispositivo conectado **se realizan en la aplicación de visualización y gestión**, con el objetivo de liberar la máxima carga computacional posible al dispositivo IoT.

Los datos que se transmiten entre la aplicación de visualización y el dispositivo siempre son de distancias y tras los cálculos mencionados anteriormente son transformados a volúmenes. Estos cálculos obviamente dependen del tipo de depósito con el cual estemos trabajando en cada momento (ver figura 63 del apartado *Storyboard*).

Para calcular el **volumen total** de un depósito de tipo **rectangular** implementamos la siguiente fórmula:

$$V = Ancho * Largo * Alto$$

que se modificaría de la siguiente forma para averiguar el **volumen ocupado** por el líquido que contenga, teniendo en cuenta que "distancia" será la distancia existente entre el techo del depósito y el nivel del líquido, obtenida por el dispositivo IoT :

$$Vo = Ancho * Largo * (Alto - distancia)$$

De la misma forma, pero para el caso de un depósito de tipo **cilíndrico** utilizaremos las siguientes fórmulas para el cálculo del volumen total y el volumen ocupado:

$$V = \pi * Radio^2 * Alto$$

$$Vo = \pi * Radio^2 * (Alto - distancia)$$

Por último, el caso más enrevesado es el del depósito con forma de **cisterna o cilindro horizontal**. En la siguiente figura se muestra un corte transversal de un depósito de este tipo, con un nivel de líquido '*h*':

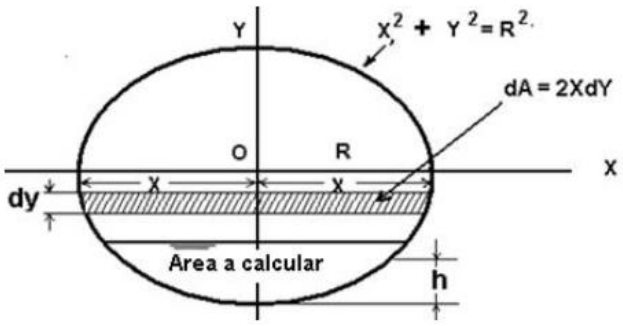


Figura 46: Corte transversal de un depósito tipo cisterna

A partir de esta figura vamos a obtener la ecuación que nos permita calcular el área ocupada en dicha base, que posteriormente nos permitirá calcular el volumen total del depósito:

$$\text{Si } x^2 + y^2 = R^2$$

$$X = \sqrt{R^2 - y^2}$$

$$dA = 2xdy, \text{por lo tanto}$$

$$dA = 2\sqrt{R^2 - y^2}dy$$

Por lo tanto:

$$A = \int dA = \int 2xdy = \int_{-R}^{-R+h} 2\sqrt{R^2 - y^2}dy$$

De la ecuación del cálculo integral:

$$\int \sqrt{a^2 - v^2}\, dv = \frac{v}{2}\sqrt{a^2 - v^2} + \frac{a^2}{2}\, arc\, sen\, \frac{v}{2} + C$$

$$A = 2\left[\frac{y}{2}\sqrt{R^2 - y^2} + \frac{R^2}{2}\, arc\, sen\, \frac{y}{R}\right]_{-R}^{-R+h}$$

$$A = 2\left[\frac{-R+h}{2}\sqrt{R^2 - (-R+h)^2} + \frac{R^2}{2}\, arc\, sen\left(\frac{(-R+h)}{R}\right) - \left(\frac{-R}{2}\sqrt{R^2 - (-R)^2}\right) + \frac{R^2}{2}\, arc\, sen - \frac{R}{R}\right]$$

Donde teniendo en cuenta que:

$$\left(\frac{-R}{2}\sqrt{R^2 - (-R)^2}\right) = 0$$

Tendremos que:

$$A = 2\left[\frac{-R+h}{2}\sqrt{R^2 - R^2 + 2Rh - h^2} + \frac{R^2}{2}\, arc\, sen\frac{-R+h}{R} + 0 - \frac{R^2}{2}\, arc\, sen - 1\right]$$

Y multiplicando el 2 que tenemos fuera del corchete nos quedaría:

$$A = \left[(-R+h)\sqrt{2Rh - h^2} + R^2\, arc\, sen\frac{-R+h}{R} - R^2\left(-\frac{\pi}{2}\right)\right]$$

$$A = \frac{\pi R^2}{2} + (h-R)\sqrt{2Rh - h^2} + R^2\, arc\, sen\, \frac{-R+h}{R}$$

Esta sería el **área de la base del cilíndro**, y por tanto para conocer el volumen tenemos que multiplicar por el largo del depósito, quedando la fórmula de la siguiente forma:

$$A = L\left[\frac{\pi R^2}{2} + (h - R)\sqrt{2Rh - h^2} + R^2\, arc\, sen\, \frac{-R + h}{R}\right]$$

Para el caso de querer conocer el **volumen total** del depósito tendríamos que tener en cuenta que:

$$h = 2 * Radio$$

Y para el caso de querer conocer el **volumen ocupado** tendremos que tener en cuenta que:

$$h = (2 * Radio) - distancia$$

Para todos estos cálculos es fundamental que tanto los datos de las dimensiones del depósito introducidos por el usuario como la distancia obtenida por el sensor siempre esten en la **misma unidad de medida** (en nuestro caso utilizamos los centímetros y obtenemos el volumen, tras la transformación pertinente, en decímetros cúbicos (dm^3) o lo que es lo mismo en Litros (L)).

Otra de las características que debemos implementar en esta alternativa es la capacidad del dispositivo IoT de **enviar un correo electrónico de manera totalmente autónoma en caso de que el nivel del depósito disminuya por debajo del nivel mínimo definido por el usuario**. Dispondremos de dos alternativas para solucionar esta funcionalidad:

- Disponer y configurar nuestro **propio servidor SMTP** en un equipo distinto.
- Utilizar un servicio web independiente denominado **IFTTT**.

Nosotros hemos decidido la opción de emplear IFTTT ya que elimina la necesidad de requerir un equipo físico independiente dedicado a ésta tarea y además nos va a permitir conocer este servicio tan interesante.

IFTTT (de sus siglas en inglés **IF This Then That**) es un servicio web que permite crear y programar eventos para automatizar diferentes tareas en Internet. Es utilizado en un amplio rango de dispositivos domésticos y sobre todo para la automatización de la publicación de contenido a través de diferentes redes sociales.

Figura 47: Elementos de IFTTT

En nuestro caso vamos a utilizar IFTTT para programar una regla que nos permita enviar un correo electrónico cuando se cumpla una condición determinada (para nosotros la condición será que el nivel del depósito disminuya por debajo del nivel mínimo). Para esto debemos acceder al sitio web de IFTTT y crear una cuenta gratuita. Después podremos crear nuestras propias recetas, y para ello debemos acceder a la sección **MyApplets > New Applet**.

Como hemos visto debemos configurar tanto la acción o evento como la reacción, ambos por separado. Para empezar debemos indicar el evento que debe cumplirse (el elemento "**This**"). Esto se realiza haciendo clic en el elemento "+ This" en la pantalla de creación de un nuevo *Applet*. Es un proceso muy intuitivo. Nos aparecerá un campo de búsqueda para seleccionar el servicio que queremos utilizar. Nosotros seleccionaremos el servicio *Webhooks*. La característica de este servicio web es que se mantiene a la espera de una petición web. Ahora tendremos que poner un nombre al evento, por ejemplo "enviar_email", y pulsar sobre la opción "Create Trigger" para terminar la creación del disparador de nuestro *Applet*.

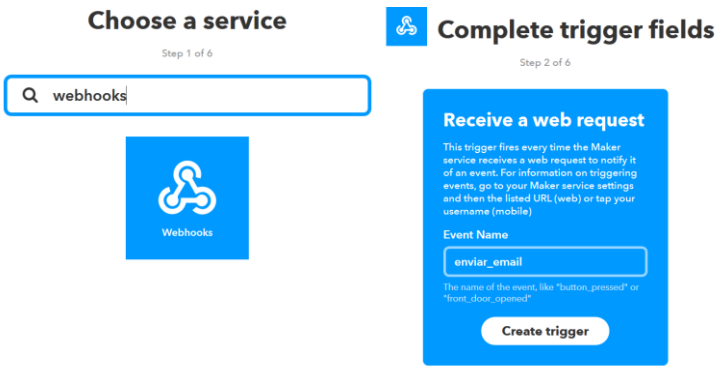


Figura 48: Configuración de la acción o evento "This" en IFTTT

Ahora toca configurar la reacción "**That**" de nuestra receta, es decir la aplicación o el servicio que queremos utilizar cuando se desencadene la acción establecida en el elemento "This". Para empezar haremos clic en "+ That" y buscamos el servicio de correo *Gmail*. Nos pedirá que conectemos con nuestra cuenta de *Gmail* que será el remitente de los correos que enviará nuestro dispositivo IoT. Ahora seleccionamos la opción "Send an email" y nos aparecerá un formulario donde tendremos que rellenar una serie de campos, de los cuales los más importantes son:

- **To Address**: Dirección de correo electrónico del destinatario.
- **Subject**: Asunto del correo electrónico.
- **Body**: Cuerpo del correo electrónico.

Podemos personalizar estos campos con los datos que posteriormente vamos a enviar en la petición *HTTP* haciendo referencia a los valores "EventName", "Value1", "Value2", "Value3" y "OcurredAt".

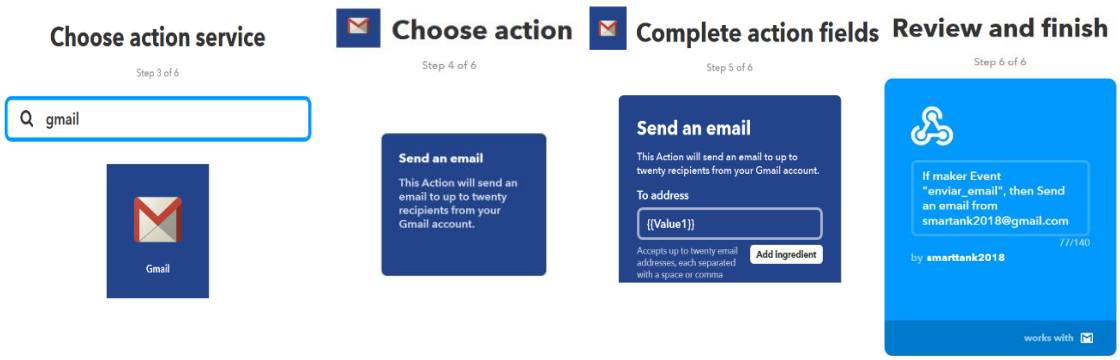


Figura 49: Configuración de la reacción o elemento "That" en IFTTT

Gracias a esta funcionalidad podemos crear un correo electrónico como este indicando como dirección destino el "Value1" que posteriormente se corresponderá con el email indicado por el usuario en la aplicación de gestión y visualización:

Figura 50: Email de notificación que envía nuestro dispositivo IoT

Por último nos aparecerá una pantalla donde veremos el resumen del *Applet* que vamos a crear, para lo cual sólo tendremos que finalizar haciendo clic en "Finish". Para ver todos nuestros *Applets* podemos acceder a la sección "My Applets".

Para acceder a este servicio que acabamos de configurar tenemos que hacer una petición *HTTP*. Para saber cómo debe ser dicha petición que activará el evento debemos dirigirnos a la dirección web https://maker.ifttt.com/ y entrar en la sección "Settings". En la ventana que nos aparece se nos muestra información relativa al servicio *Maker* en nuestra cuenta. Lo más importante es que debemos obtener la *URL* que aparece asociada a nuestra cuenta y abrirla en una nueva ventana del navegador. Nos aparecerá un sitio web donde podemos rellenar los datos de la petición *HTTP* con el nombre de nuestro evento (en nuestro caso "enviar_email") y los datos que deseemos para comprobar el correcto funcionamiento de nuestro *Applet* (sobre todo la dirección de correo de destino en el campo "Value1") haciendo clic en "Test it". Si lo hacemos comprobaremos que recibimos un correo electrónico en la dirección indicada con la configuración que hemos establecido en nuestro *Applet*. Lo importante de este paso es tener en cuenta que hemos realizado una petición *HTTP* a la *URL*:

```
1.  https://maker.iftttt.com/trigger/{nombre_evento}/with/key/{ clave_secreta}
```

La clave secreta nos aparece en el sitio web donde hemos realizado la prueba del *Applet* y el nombre del evento ya lo conocemos. Además si tenemos que enviar datos adicionales, como en nuestro caso, debemos tener en cuenta que la petición debe utilizar el método *POST* y añadir estos datos en formato *JSON* en el cuerpo de la petición. Por tanto la petición *HTTP* que debemos realizar para nuestro *Applet* quedaría de la siguiente forma:

```
1.  POST /trigger/enviar_email/with/key/PRIVATE_KEY HTTP/1.0
2.
3.  Host: maker.ifttt.com
4.
5.  Connection: close
6.
7.  Content-Type: application/json
8.
9.  Content-Length: 32
10.
11. { "value1" : "myMail@mail.com", "value2" : "02", "value3" : "03" }
```

Esta petición es la que implementaremos en nuestro dispositivo IoT enviando como valor 1 el email destino y como valor 2 el nivel mínimo en porcentaje establecido para el depósito.

6.3.2. Comunicación con una *Plataforma IoT*

La segunda alternativa que vamos a utilizar en nuestro proyecto IoT es utilizar una Plataforma IoT como herramienta de gestión y visualizacion. Para esto debemos adecuar el código del programa que se cargará en NodeMCU con el objetivo de realizar la conexión pertinente con la Plataforma IoT, pues esta comunicación requiere de una configuración específica. Por esta razón vamos a dedicar este apartado para explicar las modificaciones que hemos realizado en nuestro programa con este objetivo.

Como ya comentamos en la sección 3.3 del tercer capítulo de este documento, la *Plataforma IoT* que vamos a utilizar para nuestro proyecto es *ThingSpeak*. Tanto las características fundamentales de esta *Plataforma IoT* como las razones que nos llevan a tomar esta decisión se encuentran explicadas en la sección comentada anteriormente, por tanto aquí vamos a explicar solamente los **aspectos relativos la implementación del código correspondiente para la comunicación** entre nuestro dispositivo conectado, *NodeMCU*, y la

Plataforma IoT ThingSpeak que nos servirá como herramienta de gestión y visualización.

Comenzaremos accediendo al sitio web de ThingSpeak y crear una cuenta gratuita. Como ya comentamos en la sección correspondiente existen distintos planes de suscripción para esta Plataforma IoT, pudiendo elegir entre cualquiera de ellos según las necesidades de transmisión, el número de dispositivos que vayamos a necesitar para nuestro proyecto y por supuesto el presupuesto de que dispongamos, sin embargo ThingSpeak nos ofrece un plan gratuito totalmente funcional pero con limitaciones.

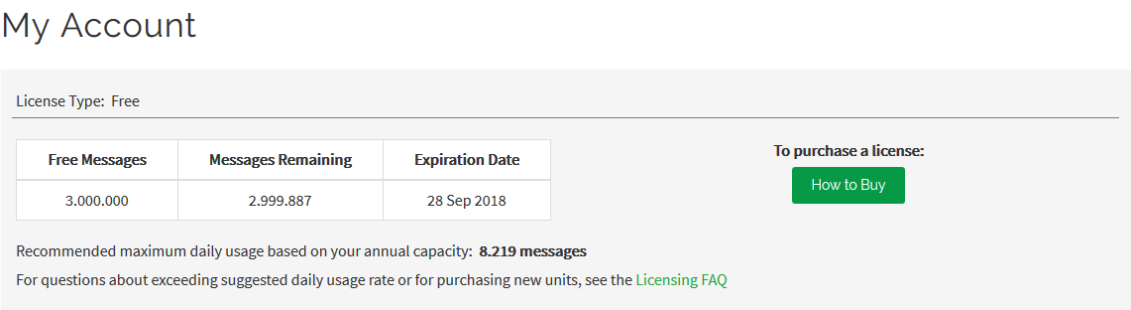


Figura 51: Características de la cuenta gratuita de ThingSpeak

Estas limitaciones no suponen problema para nuestro proyecto por lo que podemos utilizar tranquilamente esta Plataforma IoT de forma totalmente gratuita.

Ahora que ya tenemos nuestra cuenta de ThingSpeak debemos crear un canal. Un canal es una estructura ThingSpeak a la cual asociamos un proyecto. Los canales almacenan los datos que envian sus dispositivos asociados o las aplicaciones con las que trabaja. Cada canal incluye ocho campos que pueden alojar cualquier tipo de dato, además de tres campos adicionales destinados a localización y otro más para datos de estado. A partir de los datos almacenados en un canal podemos utilizar las aplicaciones de ThingSpeak para analizar, realizar cálculos y visualizar dichos datos. Podemos crear diferentes canales, o lo que es lo mismo diferentes proyectos, con una misma cuenta de ThingSpeak. Para crear un nuevo canal nos dirigimos a la sección **Channels > My Channels** y seleccionamos la opción “New Channel”.

En la pantalla donde nos redirige el sitio web debemos rellenar un formulario con los datos que personalizarán nuestro canal. No vamos a entrar a explicar en profundidad ahora esta parte, ya que lo haremos en la sección corresponiente del capítulo siguiente, pero necesitamos crear el canal para poder crear el código del programa para nuestro dispositivo NodeMCU. Por tanto por ahora solo necesitamos saber que tendremos que rellenar, al menos:

- **Nombre del canal**: Smart Tank
- **Campo 1**: Distancia
- Establecer el canal como **público** (marcar la casilla “Public”) para acceder más tarde fácilmente a los datos alojados en él.

Al pulsar sobre el botón “Save Channel” finalizará la creación de nuestro canal. Hay que añadir que todos los datos que configuran el canal se pueden modificar posteriormente en cualquier momento.

Ahora que ya hemos creado nuestro canal debemos dirigirnos a la sección API Keys donde debemos obtener un importante dato para la comunicación con nuestro dispositivo NodeMCU. Cada canal de ThingSpeak cuenta con claves para acceder al mismo. Estas claves son únicas y se generan automaticamente por la Plataforma. Según la acción que queramos realizar sobre el canal debemos utilizar un tipo de clave u otra:

- **Write API Key**: Clave que se utilizará para escribir datos en el canal. Conocer esta clave es la única manera de añadir datos al canal, por tanto es importante cuidar de que no caiga en manos ajenas. En caso de creer que nuestra clave ha sido comprometida de alguna forma (por ejemplo, si observamos que se han insertado más datos de los que se debería) podemos eliminar la clave actual y generar una nueva clave de escritura, pero tendremos que actualizar todos nuestros dispositivos con la nueva clave.
- **Read API Keys**: Clave que se utilizará para leer los datos almacenados en los distintos campos de nuestro canal. A diferencia de la clave de escritura, esta clave no es única. Pueden existir varias claves distintas para acceder a la lectura de los datos del canal. Esto permitiría, por ejemplo, el acceso

al canal a diferentes clientes, además de añadir y eliminar clientes (claves) en cualquier momento.

Además de estas claves cada canal se encuentra identificado por un número de identidad (ID) único que también se genera automaticamente cuando lo creamos.

En este momento nosotros unicamente necesitamos conocer la clave de escritura para poder enviar los datos del sensor a la Plataforma IoT, por tanto debemos copiar y guardar dicha clave.

Ahora que ya conocemos como conseguir acceso al canal podemos comenzar a implementar el código de nuestro programa para enviar datos a la Plataforma IoT. En esta ocasión vamos a omitir la explicación de cómo realizar la conexión con la red WiFi y cómo utilizamos el sensor HC-SR04 para recolectar datos, ya que se implementaría de la misma forma que ya hemos visto en el apartado anterior, por tanto solamente vamos a ver lo exclusivo de esta alternativa, es decir el envío de datos a la Plataforma IoT.

Para este proposito lo primero que debemos escribir en nuestro programa es la clave de escritura que hemos obtenido anteriormente, que declararemos como una constante de tipo cadena de caracteres en nuestro programa.

Después también tendremos que crear un objeto de la clase WiFiClient que se encargará de enviar la petición HTTP a la dirección que le indicaremos.

En la parte de la función setup de nuestro programa en este caso no tendremos que realizar ninguna inicialización para esta tarea, por lo que directamente podemos pasar a conectar con la Plataforma IoT en la función loop.

En este caso utilizaremos la función connect de la clase WiFiClient para conectar con ThingSpeak. Esta función recibirá como parametros la dirección URL a la cual queremos conectar (podemos introducirla directamente o a partir de una variable declarada anteriormente) y el puerto al cual queremos conectarnos como clientes. Si se realiza la conexión correctamente esta función nos devolvera el valor True y podremos enviar la petición HTTP.

El último paso, por tanto, sería crear y enviar la petición HTTP con el dato captado por el sensor. El código completo de todo lo que debemos implementar para esta tarea es el siguiente:

```
1.  String apiKey = "*****************"; // APIKey de escritura para el canal
2.
3.  WiFiClient client; // Objeto encargado de establecer la conexión
4.
5.  const char* server = "api.thingspeak.com";
6.
7.  void loop(){
8.
9.   // Obtenemos el dato del sensor
10.  distance = getDistance();
11.
12.  //Conectamos con la Plataforma IoT
13.  if( client.connect(server, 80) ){
14.
15.   // Creamos el cuerpo de la petición POST
16.   String postStr = apiKey;
17.   postStr += "&field1=";
18.   postStr += String(distance);
19.
20.   postStr += "\r\n\r\n";
21.
22.   // Creamos la petición HTTP
23.   client.print("POST /update HTTP/1.1\n");
24.   client.print("Host: api.thingspeak.com\n");
25.   client.print("Connection: close\n");
26.   client.print("X-THINGSPEAKAPIKEY: " + apiKey + "\n");
27.   client.print("Content-Type: application/x-www-form-urlencoded\n");
28.   client.print("Content-Length: ");
29.   client.print(postStr.length());
30.   client.print("\n\n");
31.   client.print(postStr);
32.
33.   Serial.println("Dato enviado a ThingSpeak");
34.
35.  }
36.   Client.stop();
37. }
```

Como podemos observar, en la petición que realizamos a ThingSpeak incluiremos dentro del cuerpo de la misma el dato obtenido por el sensor asociado al **campo** (en ingles “field”) 1, que en la creación de nuestro canal hemos declarado precisamente como “Distancia”. De la misma manera podríamos añadir distintos datos a los demás campos disponibles en el canal.

Además de esto también tendremos que implementar el código necesario para controlar el flujo de datos hacia la Plataforma IoT. ThingSpeak **requiere un tiempo de 15 segundos entre peticiones de escritura**, por tanto toda petición que se realice en un periodo de 15 segundos tras una petición de este tipo no

será atendida por la Plataforma y por tanto el dato no se almacenará en ella. Teniendo esto en cuenta podemos establecer un intervalo de tiempo para mantener actualizado los datos de nuestro proyecto. En nuestro caso particular vamos a enviar el dato que refleja el nivel del depósito cada 10 minutos, pero como sabemos esto dependerá de diferentes factores (características de nuestra apliación o requisitos del cliente, por ejemplo).

Para hacer esto se puede usar la misma técnica que utilizamos en la primera alternativa que hemos visto en la sección anterior, con lo que no vamos a repetirlo aquí.

En esta alternativa para utilizar una Plataforma IoT como aplicación de gestión y visualización no necesitamos almacenar ningun tipo de dato imprescindible en nuestro dispostivo, por lo que no necesitamos utilizar la memoria Flash.

Ya tenemos listo el código necesario para lograr la comunicación con ThingSpeak. Una vez hecho esto tocaría configurar la Plataforma IoT para llevar a cabo los requisitos de nuestro proyecto, pero esto lo veremos más detenidamente en la sección correspondiente del siguiente capítulo.Con esto terminamos de explicar los aspectos más importantes que hemos tenido que estudiar y solucionar para poder desarrollar el programa que se encargará de controlar nuestro dispositivo conectado.

Capítulo VII

Implementación de la Aplicación de Gestión y Visualización

En este capítulo vamos a continuar implementando otra parte fundamental de nuestro proyecto. Como se ha indicado al final del capítulo anterior, vamos a implementar dos formas distintas de **gestionar el dispositivo que llevará a cabo la monitorización del depósito**.

Comenzaremos explicando cómo hemos desarrollado una **aplicación para smartphone bajo el sistema operativo Android**. Introduciremos en primer lugar el entorno de desarrollo adecuado para esta tarea, asi como su correcta instalación y configuración del *IDE*. Después nos centraremos en explicar detalladamente cada aspecto que nos ha parecido fundamental durante su desarrollo, desde las clases implementadas hasta el diseño de la interfaz de usuario de nuestra aplicación.

Una vez terminado este apartado cambiaremos la perspectiva para mostrar cómo configurar y utilizar una **Plataforma IoT** como aplicación de gestión y visualización para un proyecto de este tipo. Así podremos comprender mejor las diferencias entre ambas alternativas, sobre todo en costes y tiempo de desarrollo.

7.1. Aplicación para Smartphone

Como ya hemos indicado anteriormente, en este caso vamos a desarrollar una aplicación para dispositivos móviles basados en el **SO Android** mediante la cual el usuario podrá tener acceso a los datos que el dispositivo conectado recopilará sobre el depósito monitorizado, asi como enviar datos de configuración a dicho dispositivo para personalizar la experiencia de uso.

Para ello, vamos a comenzar explicando el entorno de desarrollo que hemos utilizado para desarrollar la aplicación para smartphones, que en este caso se trata del *IDE Android Studio*.

Android Studio es el entorno de desarrollo integrado oficial de Google para Android basado en IntelliJ IDEA (otro IDE para el desarrollo de aplicaciones informáticas). Permite programar utilizando como lenguaje de alto nivel Java, C++ o Kotlin (desde la versión 3.0 de Android Studio) y, además de las potentes características heredadas de IntelliJ como el editor de código y distintas herramientas para desarrolladores, Android Studio ofrece muchas más ventajas para el desarrollo y compilación de aplicaciones para Android, de las cuales podemos destacar:

- Un sistema de compilación flexible basado en Gradle.
- Un emulador propio con varias funciones.
- El entorno perfecto para desarrollar aplicaciones sobre cualquier API de Android.
- Herramienta “Instant Run” que permite aplicar cambios en el código mientras la aplicación se está ejecutando, sin necesidad de compilar un nuevo APK.
- Integración de plantillas de código y GitHub.
- Gran variedad de herramientas y frameworks de testeo.

Todos los proyectos de Android Studio se estructuran en módulos que contienen archivos de código fuente y de recursos. Hay tres tipos de módulos diferenciados: módulos de aplicaciones para Android, módulos de bibliotecas y módulos de *Google App Engine*. Si seleccionamos la **Vista de Android** dentro

de la ventana **Project** en la barra de ventanas de herramientas se nos mostrarán los archivos de nuestro proyecto actual organizados en módulos. Esta vista es la más interesante ya que nos permite un rápido acceso a los archivos clave de nuestro proyecto, además de todos los archivos de compilación, como se puede ver en la siguiente figura.

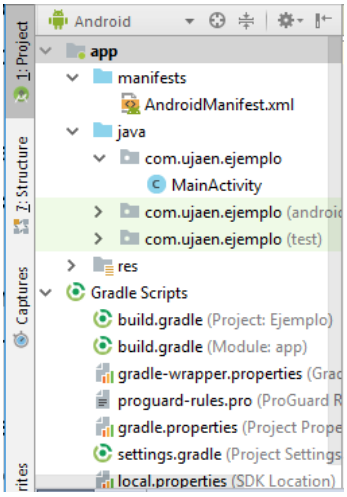


Figura 52: Vista "Android" de la ventana Project en Android Studio

Cada módulo de la aplicación contiene siempre estas carpetas:

- **Manifest**: contiene el archivo AndroidManifest en formato XML. Un archivo obligatorio en todo proyecto donde se especifica información esencial sobre la aplicación.
- **Java**: contiene los archivos de código Java de las clases que creemos en nuestro proyecto.
- **Res**: contiene los distintos recursos que usamos en nuestra aplicación, tales como diseños en XML, cadenas de interfaz de usuario para localización o imágenes.

Hay que diferenciar la representación del proyecto dentro del entorno de desarrollo y la estructura almacenada en el sistema de ficheros de nuestro ordenador, ya que no se asemejan. Para ver la estructura real que encontramos si exploramos el sistema de archivos de nuestro PC podemos seleccionar la vista **Project Files** dentro de la misma ventana **Project**.

Como hemos mencionado, Android Studio utiliza **Gradle** como sistema de compilación, pero con más capacidades específicas para Android. Los archivos de compilación son generados automaticamente por el IDE y se denominan "build.gradle". Son archivos de texto sin formato escritos en **Groovy**, un lenguaje de programación orientado a objetos implementado sobre la plataforma Java, similar a Python, Rubi o Perl. Para cada proyecto se crea un archivo de nivel superior que abarca todo el proyecto y aparte archivos independientes para cada módulo.

La interfaz de usuario del entorno puede resultar al principio un poco sobrecargada o engorrosa pero después de un tiempo de uso termina resultando intuitiva y fácil de manejar. Para facilitar este proceso de aclimatación vamos a explicar las partes más importantes de la interfaz del IDE.

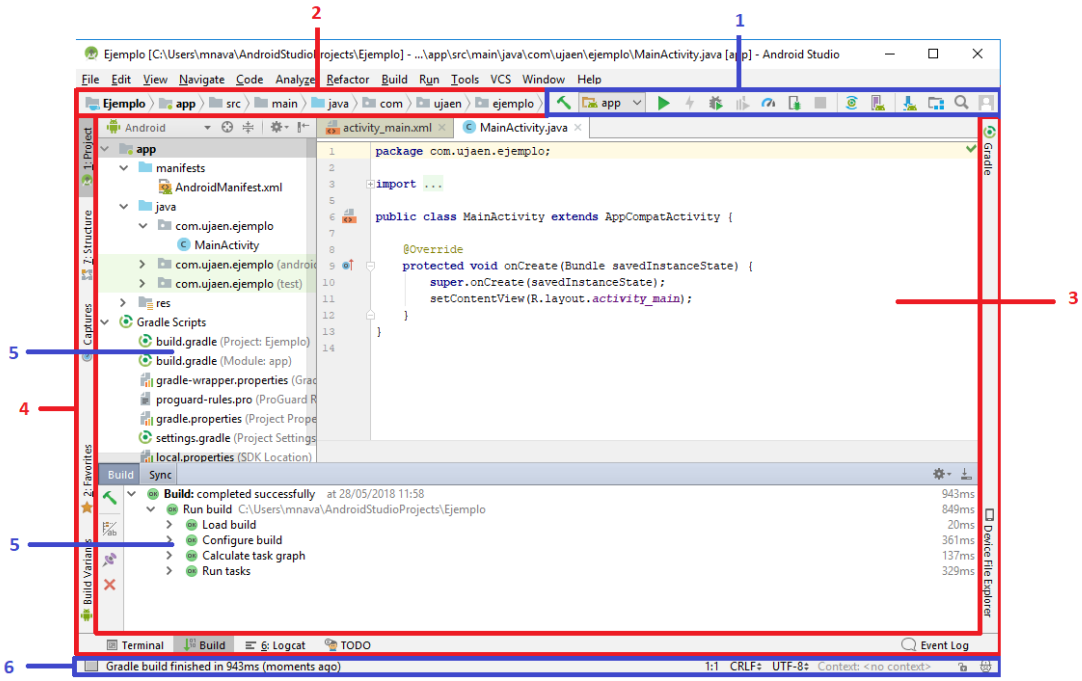


Figura 53: Interfaz de usuario de Android Studio

1. **Barra de herramientas**: desde aqui podremos acceder a una serie de acciones importantes como la compilación o ejecución de la aplicación o distintas herramientas de depuración.
2. **Barra de navegación**: como su nombre indica podremos movernos por las ubicaciones del proyecto para editar los distintos archivos que lo forman.

3. **Editor de código**: aquí es donde escribiremos el código de nuestra aplicación. Según el tipo de archivo que estemos editando el editor cambiará de aspecto, según sea un archivo .java o un archivo xml donde tendremos la vista de texto y la vista de diseño.
4. **Barra de ventanas de herramientas**: comprende todo el margen inferior y los laterales y contiene las distintas ventanas desplegables que nos ofrecerán distintas funciones dentro del IDE.
5. **Ventanas de herramientas**: permiten realizar acciones concretas y obtener información relativa a los errores, la compilación y la ejecución del proyecto.
6. **Barra de estado**: el lugar donde se muestran mensajes del IDE y donde podemos ver el estado tanto del proyecto como del entorno de desarrollo.

El proceso de instalación de este entorno de desarrollo se describe detalladamente en el Anexo B.

Para crear nuestro proyecto debemos hacer clic en la primera opción del menú de bienvenida “Iniciar un nuevo proyecto de Android Studio”.

Ahora se nos abrirá una nueva interfaz donde tendremos que empezar a personalizar nuestro proyecto:

En primer lugar nos solicitará el nombre del proyecto, que también coincidirá con el nombre que tendrá nuestra aplicación cuando la ejecutemos en un dispositivo o la publiquemos en la tienda de aplicaciones de Google Play. Nosotros a nuestra aplicación le daremos el nombre de “**Smart Tank**”.

En segundo lugar se nos pide el campo “Dominio de la compañía” que se refiere al sitio web de la empresa desarrolladora de la aplicación. Nosotros como no pertenecemos a ninguna empresa no nos interesa mucho este campo.

El siguiente campo a rellenar sí nos es más interesante, ya que define la ruta donde se ubicará nuestro proyecto. Todos los proyectos que creemos con Android Studio se guardan en la carpeta **AndroidStudioProjects** que por defecto se ubica en:

```
C:\\Users\{usuario}\AndroidStudioProjects
```

Donde `{usuario}` es el nombre de usuario en nuestro sistema. Es recomendable seleccionar esta ubicación por tema de organización de nuestros proyectos.

Posteriormente nos solicita el nombre del paquete que formará nuestro proyecto, que nos permitirá organizar las clases que creemos en él. Este campo lo rellena automaticamente Android Studio en base a los campos del dominio de la compañía y del nombre de la aplicación. Sin embargo se puede editar manualmente, aunque es algo innecesario. Lo más recomendable es dejarlo tal cual nos lo proporciona el IDE.

Por último se nos da la alternativa de poder programar nuestro proyecto en el lenguaje C++ o en Kotlin. Sin embargo nosotros vamos a utilizar el lenguaje Java por lo que dejaremos estas dos casillas desmarcadas. Pasamos ahora a la siguiente interfaz de configuración haciendo clic en “siguiente”.

Esta interfaz es bastante importante ya que nos pide información acerca de los dispositivos para los cuales deseamos programar nuestra aplicación, al mismo tiempo que debemos indicar la versión de Android con la que vamos a trabajar.

Lo primero es seleccionar el tipo de dispositivo entre las distintas opciones que se nos proporciona: Smartphones y Tablets, dispositivios “vestibles” (wearables), SmartTV, Android Auto y Android Things. A pesar de que por la naturaleza de nuestro trabajo puede parecer que la opción idónea es la de Android Things debemos recordar que esta parte del proyecto se refiere al desarrollo de una aplicación para dispositivos móviles desde la cual controlar nuestro dispositivo conectado (nuestra cosa, “thing”) por tanto no procede seleccionar dicha opcion puesto que no vamos a desarrollar con Android Studio el software que controlará nuestro dispositivo conectado, sino una aplicación para dispositivos Android desde la cual controlaremos el dispositivo. Esta opción se utiliza para desarrollar programas para plataformas o kits de desarrollo como Raspberry Pi 3, Intel Galileo, NXP Pico y algunas otras más soportadas. Por tanto nosotros en este paso marcaremos la opción de “Smartphones y Tablet”.

Además deberemos seleccionar la versión de Android sobre la que queremos programar nuestra aplicación. Esta es una decisión muy importante, ya que elegir una versión muy actual de Android supondrá limitar el número de dispositivos que pueden utilizar nuestra aplicación, es decir, supone prescindir de un buen porcentaje de mercado que podría ser cliente potencial de nuestra aplicación. Con lo cual debemos estudiar con antelación si nuestra aplicación requiere una API o versión del sistema operativo Android determinada para su desarrollo. (Por ejemplo, si quisieramos desarrollar una aplicación de realidad virtual que requiera el uso del SDK de realidad virtual de Google no tendríamos más remedio que seleccionar al menos la API 19 correspondiente con la versión de Android 4.4 “Kit Kat”). Para ayudarnos en esto, Android Studio nos ofrece un cuadro de diálogo donde podemos comprobar las funciones que añaden cada una de las distribuciones existentes de Android y el porcentaje de dispositivos que pueden acceder actualmente a dicha distribución (ver figura 54)

En nuestro caso nos bastará con la **API 15: Android 4.0.3 (IceCreamSandwich)**, a la que actualmente ya tienen acceso el 100% de los dispositivos que utilizan el sistema operativo Android, con lo cual abarcaríamos todo el mercado potencial posible si quisieramos desarrollar nuestra aplicación con fines comerciales. Aceptaremos y seguiremos con la siguiente interfaz.

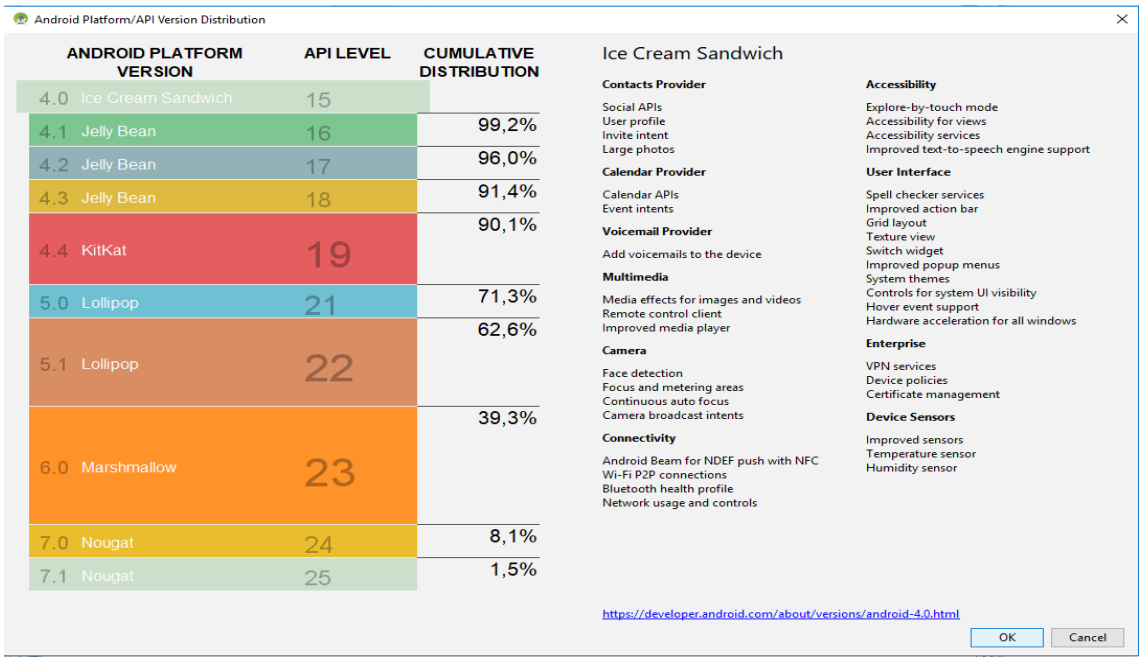


Figura 54: Las distintas APIs de Android sobre las que podemos desarrollar apps

En esta interfaz Android Studio nos preguntará con qué tipo de *Activity* queremos comenzar nuestra aplicación. Un *Activity* es el nivel más bajo que programaremos para nuestra aplicación y se corresponde con una pantalla (o vista) de nuestra aplicación. Por ejemplo, si pensamos en una aplicación de mensajería instantanea la pantalla donde podemos ver todas las conversaciones que tenemos actualmente se correspondería con un *Activity*, y cuando pulsamos sobre una conversación y nos aparece la pantalla donde podemos chatear con ese contacto sería otra *Activity* distinta.

Una *Activity* está formada por dos partes bien diferenciadas:

La **parte lógica**, que se corresponderá en nuestro proyecto con un archivo con extensión *.java* y es la clase que controlará el funcionamiento del *Activity*. El código de una clase *Activity.java* simple es el siguiente:

```
1.  package com.ujaen.ejemplo;
2.
3.  import android.support.v7.app.AppCompatActivity;
4.  import android.os.Bundle;
5.
6.  public class MainActivity extends AppCompatActivity {
7.
8.      @Override
9.      protected void onCreate(Bundle savedInstanceState) {
10.
11.         super.onCreate(savedInstanceState);
12.         setContentView(R.layout.activity_main);
13.
14.     }
15. }
```

La primera linea se corresponde con el nombre del paquete donde se encuentra la clase.

La palabra reservada "*import*" sirve en Java para importar librerias tanto propias como externas. Se utiliza seguida de la ruta al paquete o archivo que queremos agregar al proyecto.

En la línea 6 es donde se crea la clase propiamente dicha, en este caso se está creando una clase con el nombre "MainActivity" que hereda de una clase superior predefinida de Android llamada *AppCombatActivity* que añade retrocompatibilidad para versiones anteriores de Android.

Todas las clases *Activities* deben implementar obligatoriamente el método **OnCreate**, que se ejecuta cuando se crea la *Activity*. Aquí es donde se enlaza la parte lógica de la *Activity* con la parte gráfica, gracias al método **setContentView** que recibe como parámetro la ruta al archivo XML que va a mostrarse cuando se lanza la *Activity*.

La **parte gráfica** se corresponde con un archivo de formato XML que contiene los elementos que se mostrarán por pantalla (y su ubicación concreta dentro de dicha vista).

Ahora que sabemos qué es un *Activity* seleccionamos la opción más adecuada para nuestro proyecto de entre los distintos tipos que se nos da a escoger. Para nuestro proyecto basta con seleccionar el tipo “Empty Activity” que se corresponde con una *Activity* vacía similar al ejemplo que hemos visto anteriormente.

Para finalizar la creación de nuestro proyecto solamente tenemos que asignar un nombre al *Activity* que vamos a crear como principal (en nuestro caso dejamos el nombre por defecto, “MainActivity”), dejando marcadas las opciones de “Generate Layout File”, para que el IDE genere automaticamente el archivo XML que se corresponderá con la parte gráfica de nuestro *Activity* principal, y “Backwards Compatibility (AppCombat)” para que nuestra aplicación sea compatible con versiones anteriores de Android. Una vez hecho esto hacemos clic en finalizar para terminar por fin con la configuración inicial de nuestro proyecto. En este momento Android Studio nos creará una aplicación básica sobre la que desarrollaremos nuestro proyecto. El proyecto inicialmente cuenta con una clase principal denominada “MainActivity” y su parte gráfica asociada en un archivo XML denominado “activity_main”.

A partir de aquí, con todo lo necesario para desarrollar nuestra aplicación toca comenzar con su diseño. Para explicar mejor este proceso lo vamos a dividir en tres partes, atendiendo al diseño de las clases que conformarán nuestra aplicación, a los datos que manejará y a la interfaz que se mostrará y utilizará el usuario.

7.1.1. Diseño de Clases

Este apartado nos va a permitir **describir las clases que compondrán la aplicación de gestión y visualización** de nuestro proyecto, para lo cual utilizaremos un diagrama de clases. Un **diagrama de clases** no es más que la representación gráfica de la estructura de las diferentes clases y su interacción entre ellas que posteriormente será implementado utilizando un lenguaje orientado a objetos, en nuestro caso utilizaremos el lenguaje Java. Para representar el diagrama de clases vamos a utilizar el **Lenguaje Unificado de Modelado**, también conocido como **UML**, que es un estandar para describir la estructura, el comportamiento y los límites de un sistema y los objetos que contiene (Aguilar, 2015). Para conocer mejor cómo se construyen este tipo de diagramas recomendamos acudir al Anexo A

Para realizar el diagrama de clases completo de nuestra aplicación vamos a utilizar un completo editor de diagramas web: LucidChart. Aquí representaremos las clases creadas en archivos con extensión "*.java*". Estos archivos se encuentran en la ruta **App > java > com.ujaen.smart_tank** dentro de la vista *Android* de la ventana de herramientas *Project* en Android Studio.

En la siguiente página se muestra el diagrama completo de nuestra aplicación.

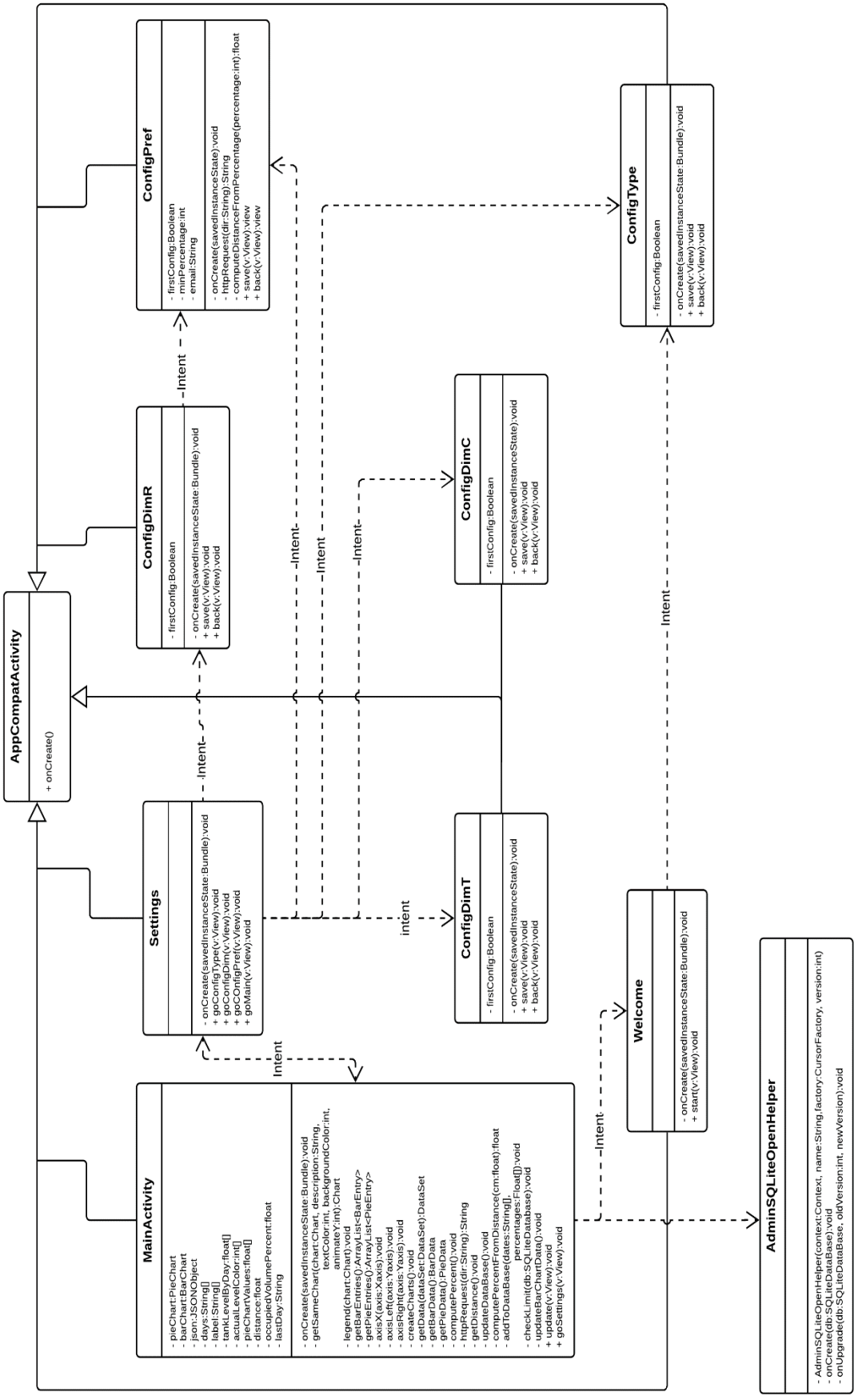


Figura 55: Diagrama UML

7.1.2. Diseño de Datos

En esta parte del diseño vamos a explicar **cómo se estructura la información almacenada en nuestra aplicación**. La información almacenada será imprescindible para el correcto funcionamiento de la aplicación y deberá poder ser recuperada al reanudar la aplicación después de su cierre o en caso de que el dispositivo móvil se apague.

En nuestro caso vamos a requerir almacenar información relativa a:

- Un **conjunto de variables** que regularán el funcionamiento del sistema.
- Los datos del nivel que presenta el depósito monitorizado a lo largo de los últimos días.

El sistema operativo Android nos facilita distintas herramientas para el almacenamiento permanente de información sensible para nuestra aplicación. Dependiendo de las necesidades de nuestro sistema podremos usar la opción que más nos interese, e incluso podremos combinar varias de ellas en un mismo proyecto de Android Studio. Algunos de los métodos que podemos utilizar para este proposito son:

- Mediante la clase **SharedPreferences**: Adecuado cuando tenemos que almacenar una cantidad limitada de datos, usualmente relativos a datos de configuración de la aplicación.
- Guardar datos en un **archivo de texto** en la memoria interna o en almacenamiento externo como una tarjeta SD.
- Utilizar una **base de datos**. Existe una herramienta nativa de Android que nos permite almacenar y organizar datos en una base de datos Open Source denominada **SQLite**. No requiere configuración, no requiere un servidor de base de datos ejecutandose en un proceso separado y se puede utilizar facilmente mediante sentencias de comandos SQL.

Para nuestro proyecto vamos a utilizar tanto la clase SharedPreferences como una base de datos personalizada gracias a SQLite.

- **SharedPreferences**

Con la clase **SharedPreferences** vamos a almacenar los datos de configuración de la aplicación y las preferencias que el usuario introducirá manualmente cuando se ejecute por primera vez la aplicación y a posteriori accediendo al menu de configuración correspondiente. Esta clase internamente nos creará un archivo XML donde guardará permanentemente toda la información que le indiquemos.

Para ello debemos obtener una referencia a un objeto de la clase SharedPreferences a través del método *getSharedPreferences* indicando como parámetros el nombre del archivo de preferencias que queremos crear/obtener y el modo de utilización de dicho archivo:

```
1.   SharedPreferences pref = getSharedPreferences("myPref", Context.MODE_PRIVATE);
```

Como vemos en nuestro caso el nombre del archivo de preferencias que utilizaremos se denomina “myPref” y el modo de utilización es *MODE_PRIVATE* que indica que el archivo “myPref” solo es accesible por nuestra aplicación.

Ahora con nuestro objeto que hace referencia al archivo gestionado por la clase SharedPreferences podemos extraer los datos del archivo de preferencias simplemente indicando el nombre del dato a extraer, un valor de retorno para el caso en que no exista un dato asociado a ese nombre (en nuestro caso en la primera ejecución de nuestra aplicación aún no existirán datos almacenados, como es lógico) y el método adecuado para el tipo de dato que queremos manejar (cadena de caracteres, numero entero, numero real, etc):

```
1.   String name = pref.getString("myName", "jesus");
```

En este ejemplo estamos recuperando el dato asociado al nombre “myName” dentro del archivo de preferencias asociado al objeto *pref* creado anteriormente. En caso de que no exista un dato asociado al nombre “myName” se asignará a la variable el valor por defecto indicado como segundo parámetro, en este ejemplo se asignará la cadena “jesus”.

Ahora bien, para modificar los datos guardados en el archivo de preferencias deberemos seguir el siguiente proceso:

1. Crear un objeto de la clase *SharedPreferences* y obtener una referencia al archivo tal y como hicimos anteriormente.
2. Crear un objeto de la clase *Editor* y asignarle la referencia al objeto de la clase *SharedPreferences* que acabamos de crear mediante el metodo *edit*() de dicha clase.
3. Utilizar el metodo asociado al tipo de variable que vamos a almacenar (*putString*, *putInt*, *putFloat*, etc) indicando como parametros primero el nombre que se va a asociar a este dato y después el propio dato que queremos almacenar.
4. Por ultimo debemos llamar al metodo *commit*() de la clase *Editor* que hará efectivos los cambios realizados en el archivo de preferencias, quedando almacenados los datos de forma permanente.

```
1.  SharedPreferences pref=getSharedPreferences("myPref", Context.MODE_PRIVATE);
2.  Editor editor=pref. edit();
3.  String name = "Pedro";
4.  editor.putString("myName", name);
5.  editor.commit();
```

En el ejemplo anterior guardaríamos la variable *name* en el archivo de preferencias denominado “myPref” asociada al nombre “myName” por lo que cuando queramos recuperar el valor de dicha variable tendremos que hacer referencia al nombre “myName” para referirnos a este dato.

En nuestro caso para la aplicación que vamos a desarrollar necesitamos almacenar los siguientes datos dentro de un archivo de preferencias denominado “configuracion” mediante la clase SharedPreferences:

Nombre	Tipo de dato	Explicación
configurado	Boolean	Indica si se ha realizado la **configuración inicial** de la aplicación, que se realiza la primera vez que se ejecuta la aplicación en el dispositivo movil.
tipo	String	Puede contener los valores 'R', 'C' o 'T' refiriendose al **tipo de depósito con el que trabajamos** (rectangular, cilindrico o cilindrico horizontal)
r_ancho	Float	Valor real del **ancho** del depósito monitorizado. Tipo rectangular.
r_largo	Float	Valor real del **largo** del depósito monitorizado. Tipo rectangular.
r_alto	Float	Valor real del **alto** del depósito monitorizado. Tipo rectangular.
c_radio	Float	Valor real que indica el **radio** del depósito monitorizado. Tipo cilíndrico.
c_alto	Float	Valor real que indica el **alto** del depósito monitorizado. Tipo cilíndrico.
t_radio	Float	Valor real que indica el **radio** del depósito monitorizado. Tipo cilíndrico horizontal.
t_largo	Float	Valor real que indica el **largo** del depósito monitorizado. Tipo cilíndrico horizontal.
min	Integer	Valor del **porcentaje mínimo** que debe alcanzar el contenido del depósito monitorizado.
email	String	El **correo electrónico** al cual se enviarán los avisos cuando el nivel del depósito sea inferior al mínimo indicado.

La mayoría de estos datos se utilizan para realizar los calculos matemáticos correspondientes dentro de la aplicación y otros para su gestión.

- **Base de datos SQLite**

También tenemos que almacenar la información relativa a cómo ha evolucionado el nivel del depósito monitorizado durante los últimos días. Para ello vamos a necesitar guardar una serie de datos dentro de una base de datos. Como hemos mencionado anteriormente vamos a utilizar para esta tarea una base de datos **SQLite**.

Lo primero que tendremos que hacer es crear una clase que herede de la clase SQLiteOpenHelper. Esta nueva clase, denominada “AdminSQLiteOpenHelper” y nos permitirá crear la estructura de nuestra base de datos y actualizar las tablas y datos iniciales. Para crear una nueva clase en Android Studio tendremos que presionar clic derecho sobre la carpeta que contiene todos los archivos con extensión “.java” de nuestro proyecto, ubicada en la ruta: **App > Java > com.ujaen.smart_tank** y seleccionar la opción **New > Java Class**. En el diálogo que nos aparecerá solamente deberemos modificar el nombre de la clase que queremos crear.

Tras hacer esto se nos creará un archivo .java con una clase básica. Ahora tendremos que hacer que nuestra clase herede de la clase SQLiteOpenHelper. Esta superclase requerirá que implementemos dos métodos: onCreate y onUpgrade. También tendremos que definir el constructor de esta nueva clase.

En el constructor simplemente debemos codificar una llamada al constructor de la superclase. Lo más importante es el método onCreate donde deberemos realizar la creación de la tabla que contendrá los datos que vamos a almacenar.

El código de nuestra clase AdminSQLiteOpenHelper quedaría así:

```
1.  public class AdminSQLiteOpenHelper extends SQLiteOpenHelper {
2.
3.      //Constructor
4.      public AdminSQLiteOpenHelper(Context context, String name,
5.                                   SQLiteDatabase.CursorFactory factory,
6.                                   int version) {
7.
8.          super(context, name, factory, version);
9.      }
10.
11.     @Override
12.     public void onCreate(SQLiteDatabase db) {
13.         db.execSQL("create table registro(fecha text primary key, porcentaje real)");
14.     }
15.
16.     @Override
17.     public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
18.
19.     }
20. }
```

Como podemos ver en el método *onCreate* vamos a realizar la sentencia encargada de crear la tabla registro que contendrá los atributos fecha y porcentaje. Así crearíamos nuestra sencilla base de datos que cumplirá perfectamente para poder recopilar y recuperar posteriormente los datos que nos permitirán representar la evolución del depósito a lo largo del tiempo.

El diagrama de Entidad-Relacion (ER) asociado a nuestra base de datos sería también muy simple, dadas las características de nuestra base de datos:

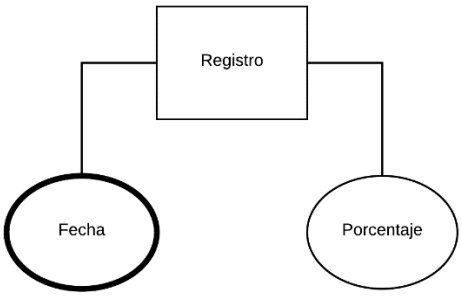


Figura 56: Diagrama Entidad-Relación de nuestra base de datos bajo SQLite

Para acceder y modificar los datos almacenados en la base de datos debemos proceder de la siguiente forma:

Para **añadir filas a la tabla** registro debemos en primer lugar crear un objeto de la clase que hemos explicado anteriormente pasandole al constructor los

parámetros *this* (contexto actual), "*admin*" el nombre de la base de datos (si no existe hasta ese momento se creará y se accederá a partir de entonces a ella por el nombre indicado), un *null* y un 1 indicando que es la primera versión de la base de datos.

Después tendremos que crear otro objeto, pero esta vez de la clase *SQLiteDataBase* invocando para esto al método *getWritableDataBase* para obtener una base de datos tanto de lectura como escritura.

Tras esto debemos crear un objeto de la clase *ContentValues* que contendrá los datos relativos a la fila que queremos insertar en la tabla de la base de datos.

Seguidamente llamaremos al método *insert* de la clase *SQLiteDataBase* pasandole como parametros el nombre de la tabla, como segundo parametro un null y como tercer parámetro el objeto *ContentValues* que hemos creado e inicializado anteriormente.

Para finalizar la escritura en la base de datos llamamos al método *close* de la clase *SQLiteDataBase*. Este proceso se entenderá mejor con el siguiente ejemplo:

```
1.  // Obtenemos referencia al administrador de SQLite
2.  AdminSQLiteOpenHelper admin = new AdminSQLiteOpenHelper(this,"admin", null, 1);
3.
4.  // Obtenemos referencia a la base de datos
5.  SQLiteDatabase bd = admin.getWritableDatabase();
6.
7.  String nueva_fecha = "2018 05 25, vie";
8.  Float nuevo_porcentaje = 28.65;
9.
10. //Creamos objeto ContentValues
11. ContentValues nuevo_registro = new ContentValues();
12. nuevo_registro.put("fecha", nueva_fecha);
13. nuevo_registro.put("porcentaje", nuevo_porcentaje);
14.
15. // Insertamos en la tabla los datos
16. bd.insert("registro", null, nuevo_registro);
17.
18. bd.close();
```

Para **realizar consultas sobre nuestra base de datos** el proceso es parecido al anterior. Debemos crear los objetos de las clases *AdminSQLiteOpenHelper* y *SQLiteDataBase* al igual que antes (en este caso vamos a leer la base de datos por lo que emplearemos el método *getReadableDataBase*) y una vez los

tengamos debemos utilizar la clase *Cursor* para obtener la referencia de la consulta y trabajar sobre las filas devueltas. Para ello inicializaremos el objeto de dicha clase con el valor devuelto con el método *rawQuery*:

```
1.  // Obtenemos referencia al administrador de SQLite
2.  AdminSQLiteOpenHelper admin = new AdminSQLiteOpenHelper(this,"admin", null, 1);
3.  // Obtenemos referencia a la base de datos
4.  SQLiteDatabase bd = admin.getReadableDataBase();
5.
6.  String fecha_almacenada;
7.
8.  // Creamos objeto de la clase Cursor y lo inicializamos con el metodo rawQuery
9.  Cursor fila = db.rawQuery("SELECT fecha FROM registro ORDER BY fecha", null);
10.
11. // Si nos devuelve al menos una fila
12. if( fila.moveToFirst() ){
13.    // Asignamos el valor de la primera fila devuelta a la variable
14.    fecha_almacenada = fila.getString( 0 );
15. }
16.
17. bd.close();
```

Como vemos en este ejemplo, una vez que hemos inicializado el objeto *Cursor* con la consulta SQL realizada sobre la base de datos utilizaremos el método *moveToFirst* para dos tareas: en primer lugar comprobar si la consulta nos ha devuelto, al menos, una fila con los atributos requeridos (ya que el método *moveToFirst* devuelve un *True* o un *False* dependiendo de si ha sido posible realizar la acción) y en segundo lugar colocar el puntero *Cursor* a la primera fila devuelta. Una vez comprobado esto podemos movernos por las demás filas devueltas facilmente gracias a los métodos *moveToNext* y *moveToPrevious* de la clase *Cursor* (también estos métodos devuelven un valor booleano para controlar si se ha llegado al límite de las filas devueltas por la base de datos). Añadir que también existe el método *moveToLast* de la clase *Cursor* similar al método *moveToFirst* pero nos servirá para apuntar a la última fila devuelta por la sentencia SQL. Una vez que tengamos nuestro objeto *Cursor* apuntando a la fila indicada podremos acceder a los atributos devueltos por la misma mediante los métodos *getString*, *getInt*, *getFloat*, etc. Dependiendo del tipo de dato que queremos recuperar, e indicando como parámetro del método correspondiente el índice del atributo al cual queremos acceder dentro de la fila devuelta. Por ejemplo, si realizamos la siguiene consulta a la base de datos:

```
1.  SELECT fecha, porcentaje FROM registro ORDER BY fecha
```

El objeto de la clase *Cursor* apuntará a las filas devueltas, y cada fila contendrá dos atributos ordenados: “fecha” y “porcentaje”. Por tanto si queremos acceder al primer atributo debemos utilizar el método *getString* e indicar como índice un 0, refiriendonos al primer atributo devuelto por cada fila:

```
1.  fila.getString(0);
```

Y en caso de querer acceder al atributo “porcentaje” debemos utilizar el método *getFloat*, ya que este atributo es de tipo número real, e indicar como índice un 1:

```
1.  fila.getFloat(1);
```

De esta manera, si realizásemos una consulta con más atributos podríamos acceder a ellos de igual forma, atendiendo únicamente al tipo de variable que guarda ese atributo y utilizando los índices como si estuvieramos accediendo a un vector.

Por otro lado, si quisieramos **actualizar datos de nuestra base de datos** se procedería de manera similar al proceso de insertar datos, pero en esta ocasión utilizaríamos el método update de la clase *SQLiteDataBase* con el cual construiríamos la sentencia SQL añadiendo la cláusula *where* para indicar aquella/s fila/s que queremos actualizar y pasando los nuevos datos en un objeto de la clase *ContentValues* tal y como hicimos anteriormente para insertar nuevas filas en la tabla. Por ejemplo:

```
1.  // Obtenemos referencia al administrador de SQLite
2.  AdminSQLiteOpenHelper admin = new AdminSQLiteOpenHelper(this,"admin", null, 1);
3.
4.  // Obtenemos referencia a la base de datos
5.  SQLiteDatabase bd = admin.getReadableDataBase();
6.
7.  String fecha_almacenada;
8.
9.  // Creamos objeto de la clase Cursor y lo inicializamos con el metodo rawQuery
10. Cursor fila = db.rawQuery("SELECT fecha FROM registro ORDER BY fecha", null);
11.
12. String fecha = "2018 05 16, mie";
13. Float nuevo_porcentaje = 80.56;
14.
15. // Creamos el objeto ContentValues
16. ContentValues nuevo_registro= new ContentValues();
17. nuevo_registro.put("fecha", fecha);
18. nuevo_registro.put("porcentaje", nuevo_porcentaje);
19.
20. // Actualizamos los datos en la base de datos
21. int cant = bd.update("registro", nuevo_registro, "fecha=" + fecha, null);
22.
23. // cant = numero de filas actualizadas
```

```
24.
25.  bd.close();
```

El método *update* requiere como parámetros en este orden: el nombre de la tabla que queremos actualizar, el objeto *ContentValues*, la cláusula *where* y en el último parámetro podemos pasar un vector de tipo *String* si necesitamos indicar varias sentencias *where*.

Por último, para **eliminar filas de una tabla** de nuestra base de datos deberemos utilizar el método delete de la clase *SQLiteDataBase*. Este método se utiliza de un modo semejante al método *update* que acabamos de ver. Tan solo deberemos indicar sus parámetros: el nombre de la tabla donde queremos eliminar las filas, la cláusula *where* y, en caso de ser necesario, un vector de *String* que contenga distintos parámetros para la cláusula *where*:

```
1.   // Obtenemos referencia al administrador de SQLite
2.   AdminSQLiteOpenHelper admin = new AdminSQLiteOpenHelper(this,"admin", null, 1);
3.
4.   // Obtenemos referencia a la base de datos
5.   SQLiteDatabase bd = admin.getWritableDataBase();
6.
7.   String fecha_eliminar = "2018 05 22, mar";
8.
9.   // Eliminamos la entrada de la base de datos
10.  int cant = bd.delete("registro", "fecha=" + fecha_eliminar, null);
11.
12.  bd.close();
```

Al igual que ocurre con el método *update*, al borrar filas de la tabla el método nos devuelve un entero que indica el número de filas que se han eliminado.

7.1.3. Diseño de la interfaz de usuario

El diseño de la interfaz de usuario es un **aspecto fundamental** dentro del desarrollo de cualquier aplicación o sistema informático. Ofrecer al cliente una interfaz simple e intuitiva, pero a la vez completa, es esencial para el éxito de la aplicación.

En nuestro caso, lo más importante es que el usuario de la aplicación debe entender en cada momento que acción puede realizar y, sobre todo, poder visualizar fácilmente el estado actual de su depósito y su evolución durante los últimos días, ya que este es el propósito final de nuestra aplicación Android.

Para esta tarea hemos utilizado **gráficos** que permiten a cualquier usuario, sin importar su nivel de conocimientos tanto a nivel informático como matemático, conocer de un simple vistazo y de manera intuitiva estos datos para su valoración y uso.

Concretamente hemos decidido utilizar tanto un **gráfico circular** para representar el valor actual del nivel del depósito, como un **gráfico de barras** para representar la evolución del nivel durante los últimos días.

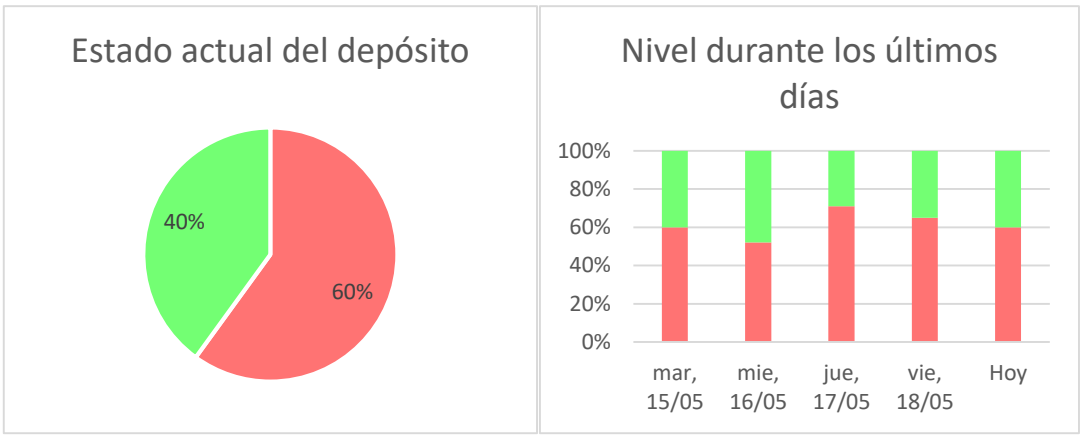


Figura 57: Tipos de gráficos utilizados en Smart Tank

Por otro lado, el **uso de colores** es igual de importante para esta tarea, por tanto hemos decidido utilizar una paleta de colores en nuestros gráficos adecuada para nuestra tarea.

Hemos utilizado dos colores (ver figura 58) bien diferenciados para indicar el porcentaje libre (representado con un color verde) y el porcentaje ocupado (representado por un color rojo) de nuestro depósito. En ambos tipos hemos utilizado una gama de color suave que resulte cómodo de visualizar.

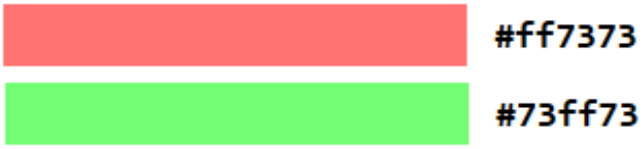


Figura 58: Paleta de colores utilizada en los gráficos de Smart Tank

Para diferenciar la representación de los datos frente a las **acciones** que el usuario puede ejecutar dentro de la aplicación hemos representado los botones de la aplicación que representan dichas acciones de una forma radicalmente

distinta. Para esto hemos utilizado un diseño con un color azul que se identifica claramente dentro de la interfaz, como se puede comprobar en la siguiente figura:

Figura 59: Diseño de los botones de Smart Tank

Además de los aspectos que ya hemos comentado es importante establecer una guia de estilo para los **textos** de nuestra aplicación, que ayudará a que todos los elementos de la misma tengan coherencia y formen una armonía agradable para el usuario.

En concreto nosotros vamos a diferenciar entre los títulos y el texto base de nuestra aplicación:

- Títulos:
  - Tipo de fuente: **Quicksand-Bold.**
  - Tamaño de letra: **30 sp** (**Scale-independent Pixels**, unidad propia de Android que se adapta al tamaño de pantalla del dispositivo de manera que el tamaño de la fuente especificado se vea igual independientemente del tamaño del dispositivo en el cual se ejecute la aplicación).
- Texto base:
  - Tipo de fuente: **Quicksand-Regular**.
  - Tamaño de letra: **14 sp**.

Otro de los aspectos fundamentales que entran en el concepto del diseño de la interfaz de usuario son los **mensajes** de información que se muestran al usuario. Suelen ser elementos dinámicos que se muestran momentaneamente cuando se cumple alguna condición, normalmente sujeta a la acción del usuario sobre el sistema.

Su función es mostrar feedback al usuario sobre las acciones que realiza en la aplicación. Por ejemplo, cuando el usuario modifica un parámetro de

configuración el sistema muestra un mensaje indicando si se ha realizado correctamente la modificación o por el contrario ha sucedido algún error en el proceso.

En nuestro caso para mostrar este tipo de mensajes vamos a utilizar la clase *Toast*. Esta clase nos permite lanzar notificaciones sencillas que aparecerán en pantalla durante un breve periodo de tiempo.

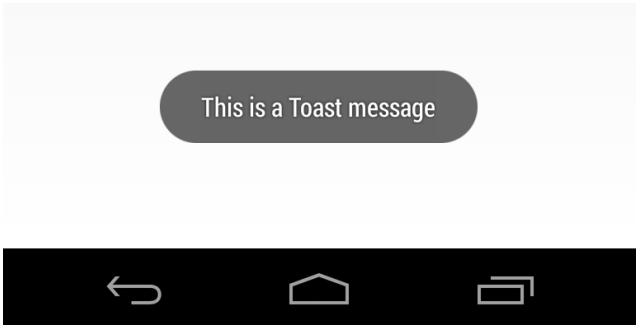


Figura 60: Ejemplo de notificación Toast

Otro aspecto importante del diseño de la interfaz de usuario es el uso de **iconos**. Normalmente se emplean metáforas para el diseño de los iconos que se van a emplear en la aplicación a desarrollar. Una metáfora es una figura retórica por la cual una realidad o concepto se expresan por medio de otra realidad o concepto diferentes que lo representan y con la cual guarda cierta relación de semejanza. Es decir representar una acción con una imagen que nos de a entender qué podemos realizar mediante la acción que desencadena dicha imagen.

En nuestro caso hemos creado un icono cuyo principal propósito es representar nuestra aplicación ante los usuarios, por lo cual lo hemos utilizado como logo principal de la aplicación, teniendo las siguientes atribuciones:

- Aparecerá en la lista de aplicaciones instaladas en el dispositivo movil, además de en la pantalla principal.
- Permite que los usuarios identifiquen rápidamente la aplicación en la tienda Google Play en caso de ser publicada.
- Crea una primera impresión de la aplicación.

El diseño del logo que vamos a utilizar para nuestra aplicación es el siguiente:

Figura 61: Logo de nuestra aplicación Smart Tank

Una vez diseñada la imagen que utilizaremos como logo utilizaremos la herramienta **Image Asset Studio** del entorno de desarrollo, la cual nos facilitará la tarea de adaptar nuestra imagen en formato .*png* para su uso como icono principal de la aplicación.

Esta herramienta generará un conjunto de iconos de distintas resoluciones para cada densidad de pantalla (en concreto: mdpi, hdpi, xhdpi, xxhdpi y xxxhdpi). Estos recursos se guardarán dentro de la ruta Así en tiempo de ejecución el sistema operativo Android utilizará la imagen apropiada según la densidad de píxeles del dispositivo donde se ejecuta la aplicación.

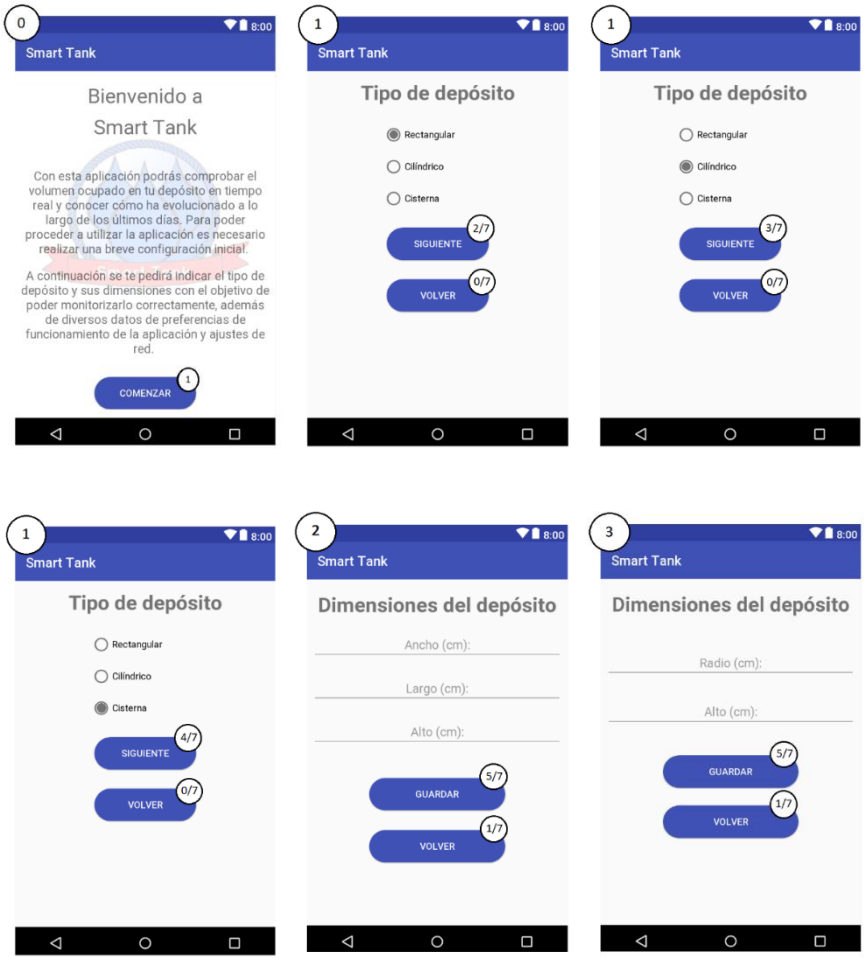
- **Storyboard**

La mejor manera de mostrar el aspecto final que tendrá nuestra aplicación una vez que sea implementada es a través de un **Storyboard**. El Storyboard es una técnica de diseño que nos permite comprobar cómo se comportará un sistema gracias a una serie de ilustraciones que se enlazarán entre sí para crear la secuencia completa que será programada posteriormente. Nos permite comprobar que el diseño de las diferentes pantallas que formarán nuestra aplicación es útil y funcionará correctamente antes de llevar a cabo su implementación, asi como detectar posibles errores y posibles mejoras en la organización.

Hay distintas formas de realizar un Storyboard, desde las más básicas con simple papel y lapiz hasta las más avanzadas mediante software especializado para ello. En nuestro caso vamos a valernos del propio entorno de desarrollo

Android Studio, ya que nos aporta la funcionalidad de visualizar los archivos *XML* que contienen la parte gráfica de los Activities que forman parte de nuestra aplicación antes de implementar la lógica de la clase que crearemos en el archivo con extensión *.java* asociado, tal y como explicamos en el apartado 7.1.1 Diseño de Clases.

El Storyboard que representa la secuencia de interacción del usuario con nuestra aplicación es el siguiente:



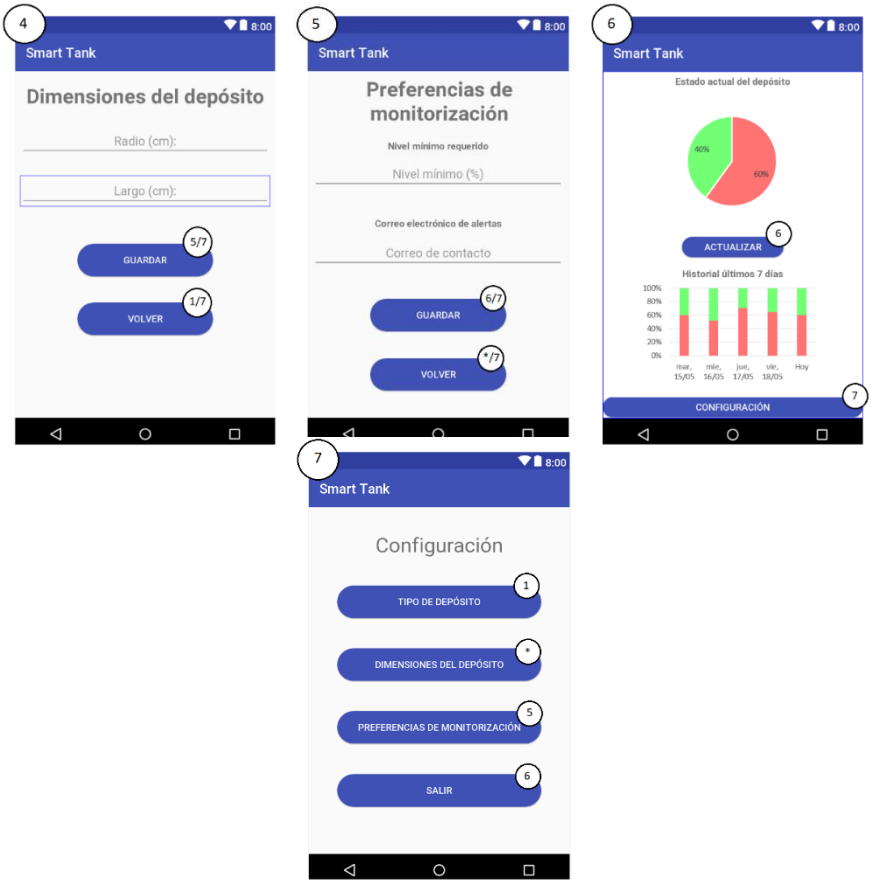


Figura 62: Storyboard de la aplicación Smart Tank

Para facilitar aún más la comprensión de la secuencia de las pantallas de nuestra aplicación vamos a explicarlo detalladamente teniendo como referencia el Storyboard.

0: Pantalla de bienvenida, se muestra la primera vez que se ejecuta la aplicación en el dispositivo movil e introduce al usuario la configuración inicial que se va a llevar a cabo. De fondo se muestra una marca de agua con el logo de nuestra aplicación. Desde aquí sólo podemos pasar a la pantalla número 1.

1: Selección del tipo de depósito, es el primer dato que se pide al usuario ya que nuestra aplicación estará preparada para calcular el nivel de volumen ocupado para tres tipos distintos de depósitos, atendiendo a su forma:

- Tipo rectangular (A)
- Tipo cilíndrico (B)
- Tipo cisterna o cilindro horizontal (C)

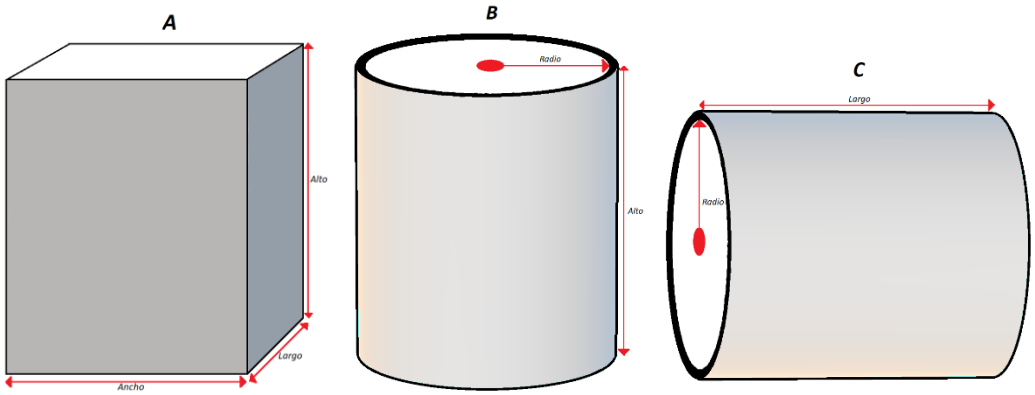


Figura 63: Tipos de depósitos soportados en Smart Tank

Desde este punto podemos pasar a tres pantallas diferentes (2, 3 o 4), dependiendo del tipo de depósito seleccionado, si pulsamos sobre el botón "siguiente" pero siempre y cuando estemos en el proceso de configuración inicial, ya que esta pantalla también va a ser accesible a través del menú de configuración de la aplicación y si llegamos a ella a través de este menú cambiará un poco tanto el aspecto visual de la pantalla como su comportamiento, puesto que el boton "siguiente" pasará a mostrar el texto "guardar", los textos de los campos a rellenar cambiarán a mostrar los datos guardados actualmente en la memoria del dispositivo (para ayudar a comprobar si es necesaria su modificación) y en lugar de avanzar a la pantalla siguiente que corresponderia en la configuración inicial (2, 3 o 4) volveríamos al menú (correspondiente a la pantalla 7 del Storyboard). Además se mostraría una notificación *Toast* para indicar al usuario que se han guardado correctamente las modificaciónes.

Esto se representa en el Storyboard con el símbolo "2/7" que indica, en el primer número antes de la barra separadora, la pantalla a la que pasaríamos si nos encontramos en el proceso de configuración inicial y, en el segundo número tras la barra separadora, la pantalla de destino si no estamos en la configuración inicial (dicho de otro modo, si hemos accedido a esta pantalla a través del menú de configuración (7)). Esta nomenclatura aparece varias veces a lo largo del Storyboard y siempre tiene el mismo significado.

2: Dimensiones del depósito (Rectangular), en esta pantalla deberemos introducir las dimensiones propias de un depósito del tipo rectangular: ancho, largo y alto. Todas ellas, tal y como se indica en los propios campos que debe

introducir el usuario, deben introducirse en la unidad de medida *centímetros*. Esto es importante para realizar los cálculos oportunos que la aplicación necesita a partir de las dimensiones que se recogen aquí.

A partir de aquí podremos avanzar hasta la pantalla de preferencias de monitorización (5) o volver a la pantalla de selección del tipo de depósito. Esto en caso de encontrarnos en la configuración inicial, ya que al igual que ocurría en la pantalla anterior, si accedemos a esta pantalla desde el menú de configuración de la aplicación, en lugar de esta secuencia tendremos la opción de modificar los datos de dimensiones y volver al menú de la aplicación (7).

3: Dimensiones del depósito (Cilíndrico), esta pantalla es similar a la anterior. La única diferencia son los datos de dimensiones que se le pide al usuario. En este caso según el tipo de depósito seleccionado previamente se requiere introducir el radio de la base del cilindro y la altura del mismo (ver figura 63).

4: Dimensiones del depósito (Cisterna o cilindrico horizontal), al igual que la pantalla anterior la única diferencia con la pantalla número 2 son los datos a introducir por el usuario. En este caso se requiere tanto el radio de la base del cilindro como el largo del mismo (ver figura 63).

5: Preferencias de monitorización, este es el último paso de la configuración inicial. Se le pedirá al usuario un nivel mínimo para el depósito y un correo electrónico. Ambos datos están relacionados, ya que serán enviados al dispositivo conectado para detectar si en algun momento el nivel de volumen ocupado por el líquido disminuye por debajo del nivel introducido por el usuario (por ejemplo, si el usuario introduce como mínimo un 15% el dispositivo detectará cuando el volumen ocupado en el depósito sea inferior a ese porcentaje) con el objetivo de enviar una notificación por correo electrónico que informará al usuario de este hecho sin necesidad de que esté ejecutandose en primer o en segundo plano la aplicación movil en ese momento. Este es un requisito que especificamos en el apartado 4.2.1.

Una vez introducidos los datos mediante el botón “guardar” accederemos a la pantalla principal de nuestra aplicación y se le indicará al usuario que se ha

completado la configuración inicial mediante una notificación *Toast*, a menos que accedamos desde el menú de configuración que en tal caso volveriamos a dicho menú una vez guardados los cambios. Si estamos en la configuración inicial y presionamos el botón “volver” regresaríamos a la pantalla determinada dependiendo del tipo de deposito que hallamos seleccionado previamente (es decir, a la número 2, 3 o 4). Esto se indica en el Storyboard mediante el icono ‘*’. Si accedemos desde el menú de configuración volveríamos a dicho menú sin guardar las modificaciones introducidas en los datos.

6: Pantalla principal, esta es la pantalla donde inicia nuestra aplicación una vez que se halla completado correctamente la configuración inicial. Cuando el usuario ejecute la aplicación en su dispostivo movil aparecerá esta pantalla principal, es la pantalla más importante de nuestra aplicación ya que en ella se nos mostrará la información del nivel actual del depósito (la cual se actualizará automaticamente al aparecer la pantalla) representada en un gráfico circular y la información relativa a la evolución del nivel durante los últimos 7 días representada en un gráfico de barras, tal y como hemos explicado anteriormente en este apartado.

Estos datos se pueden actualizar manualmente presionando el botón “actualizar”. Además desde aquí podemos acceder al menú de configuración de la aplicación presionando sobre el botón “configuración”. Además ambas gráficas serán interactivas, pudiendo el usuario interaccionar con ellas. Por ejemplo, en el gráfico de barras el usuario podrá desplazarse de izquierda a derecha para ver todos los días almacenados en la base de datos y seleccionar un día concreto de los mostrados.

7: Menú de configuración, desde aquí podremos acceder a todas las configuraciones que hemos establecido durante la configuración inicial pudiendo tanto visualizar los parámetros que están establecidos actualmente como modificarlos según nuestras necesidades en cualquier momento.

7.1.4. Comunicación con el dispositivo conectado

Ahora vamos a ver cómo podemos crear y enviar peticiones *HTTP* a través de Internet para comunicarnos con el dispositivo *IoT*. Para ello debemos realizar una serie de pasos:

En primer lugar debemos activar el acceso a Internet para la aplicación. Las aplicaciones Android necesitan poseer el permiso adecuado para acceder a Internet, por tanto nosotros como desarrolladores debemos asignarle manualmente este permiso.

Para ello debemos dirigirnos al archivo de manifiesto del proyecto, ubicado en la ruta **App > manifest > AndroidManifest.xml** e introducir la siguiente etiqueta:

```
<uses-permission android:name = "android.permission.INTERNET" />
```

Ahora que ya tenemos acceso a Internet desde nuestra aplicación tendremos que tener en cuenta que en el sistema operativo Android no es posible ejecutar operaciones de red en el hilo principal, por lo tanto tendremos que crear y ejecutar un hilo independiente para estas tareas. Existen diversos métodos para implementar hilos secundarios. Nosotros hemos creado el hilo de ejecución donde se producirán las operaciones de red mediante la clase *Thread*:

```
1.  Thread tr = new Thread(){
2.
3.      @Override
4.      public void run() {
5.          String respuesta = httpRequest(“http://192.168.1.105/nivel_actual”, 1);
6.
7.          if (respuesta.length() == 0) { // Si no se reciben datos
8.              Log.e("Error ", "No se han recibido datos");
9.          } else {
10.             // Procesamos la respuesta recibida
11.
12.             });
13.         }
14.     }
15. };
```

Una vez que creamos el hilo debemos lanzarlo para ejecutar el código contenido en él. Esto es tan simple como llamar al método *start* de la clase *Thread*:

```
1.  tr.start();
```

Sin embargo, para **asegurar el correcto funcionamiento** del hilo de ejecución principal de nuestra aplicación debemos forzar al hilo principal para que espere hasta que el hilo que acabamos de crear no termine sus tareas. Este tiempo puede variar puesto que depende de la conexión a Internet. Para esto podemos valernos del método *join* de la misma clase *Thread* cuya función es precisamente ésta que hemos mencionado. Para utilizar éste método es obligatorio incluirlo dentro de una estructura *try-catch* para prevenirnos de posibles excepciones. El código para implementar esta característica quedaría asi:

```
1.  try {
2.    tr.join();
3.  }catch (Exception e){
4.    //Tratamiento de excepciones
5.  }
```

Por tanto la petición http se ejecuta dentro del hilo que hemos creado y mientras tanto la aplicación espera a que termine, ya que necesita de los datos que se piden al dispositivo para continuar con su ejecución. En caso de no poder producirse la conexión con el dispositivo se le notificaría al usuario mediante una notificación y los datos que se muestran en la aplicación no se actualizarían.

En concreto la petición *http* la realiza, en nuestro caso, el método *httpRequest* que recibe como parámetros por un lado la *URL* que contiene la dirección del dispositivo conectado y la acción que queremos realizar, y por otro lado un identificador del método *http* asociado a esta acción (1 = GET, 2 = PUT). Así nuestro método podrá configurar una conexión a la dirección indicada y devolver la respuesta como una variable de tipo *String*:

```
1.  public String httpRequest(String dir, int method){
2.    URL url = null;
3.    StringBuilder result = new StringBuilder();
4.    int responseCode = 0;
5.    String line = "";
6.
7.    try{
8.     url = new URL(dir);
9.     HttpURLConnection connection = (HttpURLConnection)url.openConnection();
10.
11.   if( method == 1 )
12.    connection.setRequestMethod("GET");
13.   else if ( method == 2 )
14.    connection.setRequestMethod("PUT");
15.
16.   responseCode = connection.getResponseCode();
17.
18.   if( responseCode == HttpURLConnection.HTTP_OK ){
```

```
19.      InputStream in = new BufferedInputStream(conexion.getInputStream());
20.      BufferedReader reader = new BufferedReader(new InputStreamReader(in));
21.
22.      while( (line=reader.readLine())!=null ){
23.         result.append(line);
24.      }
25.    }
26.
27.    connection.disconnect();
28.
29.    }catch (Exception e){
30.      //Tratamiento de excepciones
31.    }
32.
33.    return result.toString();
34.  }
```

Como podemos ver la instancia del objeto *HttpURLConnection* se crea utilizando el método *openConnection* de la clase *URL*. Este método solamente instancia el objeto encargado de establecer la conexión, pero no la realiza en ese momento.

Para establecer la conexión configurada y ejecutar la petición debemos utilizar alguno de los siguientes métodos de la clase *HttpURLConnection*: *connect*, *getResponseCode*, *getInputStream* o *getOutputStream*. En nuestro caso utilizamos el método *getResponseCode*, que a su vez nos devuelve el código asociado a la conexión y, en caso de establecerse la conexión correctamente (es decir, si obtenemos como código de respuesta 200, que coincide con la constante *HTTP_OK* de la clase *HttpURLConnection*), leemos la respuesta y la asignamos a una variable de tipo *String*, que es el valor retornado por el método *httpRequest*.

En caso de que el método *http* que queramos utilizar conlleve el envío de datos a través del cuerpo de la petición. Para esto, primero debemos llamar al método *setDoOutput* del objeto *HttpURLConnection* pasandole como parámetro el valor *True*. Tras este paso ya podremos incluir en nuestra petición los valores necesarios que deseemos enviar:

```
1.  // Creamos u obtenemos los datos a enviar
2.  String myData = "min=15&email=mnd00003@red.ujaen.es";
3.
4.  // Activamos la escritura del cuerpo de la peticion
5.  connection.setDoOutput(true);
6.
7.  // Escribimos los datos
8.  connection.getOutputStream().write(myData.getBytes());
```

Como vemos, en nuestro caso estaríamos enviando la información relativa al nivel mínimo del nivel del depósito y el email al cual enviar las notificaciones.

Ahora que sabemos cómo establecer la comunicación entre nuestra aplicación ejecutada en nuestro dispositivo Android y el dispositivo de desarrollo que vamos a utilizar en nuestro proyecto nos surje una última barrera que tendremos que solucionar.

Como hemos visto para establecer la conexión con el dispositivo debemos conocer la dirección (la URL) perteneciente a nuestro dispositivo. Si estamos trabajando en nuestra propia red de area local a través de WiFi (cabe recordar que nuestro kit de desarrollo accede a la red gracias a su conectividad WiFi), es decir lo que se conoce como una WLAN, podremos acceder fácilmente a nuestro dispositivo mediante la dirección IP privada que nuestro router le asigna, ya sea automaticamente (si utilizamos un servidor DHCP o nuestro router proporciona esta funcionalidad) o manualmente asignandole una IP estática. Por ejemplo, si el router le asigna a nuestro dispositivo la dirección IP privada 192.168.1.104 para acceder al nivel actual del depósito con el URI que hemos explicado anteriormente tendríamos que realizar una petición http a la dirección:

```
http://192.168.1.104/nivel_actual
```

Ahora bien, el problema llega cuando queremos acceder al dispositivo desde fuera de nuestra red puesto que la IP privada que le asigna el router solamente es accesible desde dentro de nuestra red. Para solucionar esto deberemos conocer la dirección IP pública de nuestra puerta de enlace con la red, es decir nuestro router. Si conocemos esta dirección podemos acceder a nuestra red si permitimos el acceso a través del firewall de nuestro router (en caso de que nuestro router cuente con esta característica, algo habitual en casi todos los routers proporcionados por los distintos proveedores de Internet).

Sin embargo no nos soluciona del todo el problema, ya que la dirección IP pública de nuestro router la proporciona el proveedor de Internet con el cual contratamos el servicio y casi siempre (a menos que se especifique lo contrario en nuestro contrato con el proveedor) es una dirección dinámica, lo que significa que cambia

automáticamente cada cierto tiempo. Por tanto si conocemos nuestra IP pública a día de hoy dentro de un mes esta IP cambiará por lo que ya no nos será de utilidad, un mes después volverá a cambiar y así sucesivamente.

Para solucionar esto no nos queda más remedio que utilizar servicios que asignan nuestra IP pública a un nombre de dominio. Esto nos proporciona dos ventajas:

- Permite utilizar un **nombre más legible** que una dirección *IP*.
- Nos proporcionan un software que podemos instalar en un equipo que tengamos conectado a nuestra red que se encargará de **comprobar automáticamente si nuestra IP pública cambia** y sincronizar automáticamente la nueva dirección con nuestro dominio.

Existen varios servicios de este tipo, el más popular es DynDNS que antes ofrecía este servicio de forma gratuita pero hoy en día requiere una suscripción de pago. Por tanto nosotros hemos optado por utilizar otro similar a DynDNS y que sí se ofrece de forma gratuita con algunas restricciones que no nos repercuten negativamente para el desarrollo de nuestro proyecto, se trata del servicio No-IP.

Para utilizar No-IP tan solo debemos registrarnos en su sitio web con un correo electrónico, lo que nos dará acceso a una cuenta gratuita. Una vez hecho esto tendremos la opción de crear hasta 3 “hostnames” (nombres de dominio vinculados a una *IP* pública) seleccionando entre un número limitado de dominios y deberemos confirmar cada uno de nuestros “hostname” cada 30 días. En caso contrario el “hostname” dejará de ser accesible hasta que lo confirmemos manualmente (estas son las limitaciones de la cuenta gratuita, que desaparecen si compramos una suscripción anual por 49.99€).

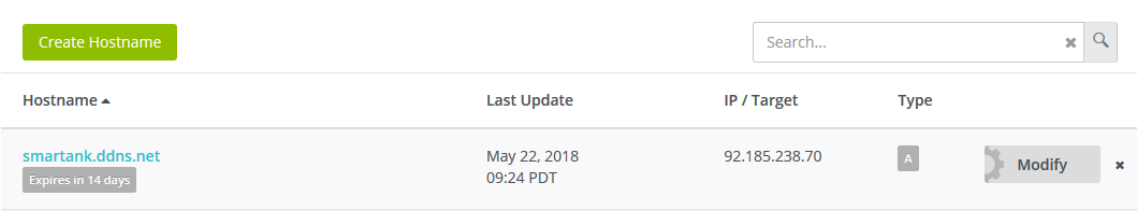


Figura 64: Nombre de dominio asociado a nuestra IP pública en No-IP

Nosotros únicamente deberemos crear un hostname y vincularlo a nuestra *IP* pública. Para esta tarea No-IP nos proporciona un software denominado **Dynamic DNS Update Client (DUC)** que debemos instalar en un equipo que se encuentre conectado a nuestra red. En este programa introduciremos los datos de nuestra cuenta de No-IP y seleccionaremos los "hosts" que queremos vincular, de esta manera DUC se encargará de vincular el nombre de dominio que hallamos seleccionado con la *IP* pública que tenga asignado nuestro router en cada momento.

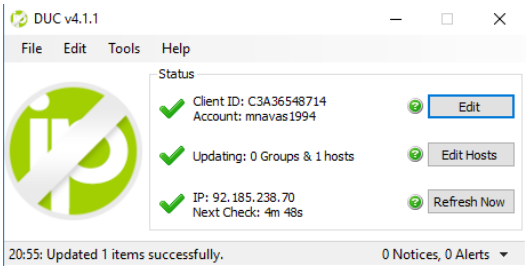


Figura 65: Interfaz de Dynamic DNS Update Client (DUC) de No-IP

El último paso que necesitamos realizar para conseguir acceder a nuestro dispositivo desde cualquier lugar con acceso a Internet es crear una regla en nuestro router para que las peticiones que lleguen al mismo se redireccionen a la IP privada asignada a nuestro dispositivo conectado. Esto se consigue mediante la redirección de puertos, una característica común de cualquier router a día de hoy que nos permite que un cliente externo a la red tenga acceso a una direccion IP privada de la red.

Es necesario por tanto que la IP privada asignada a nuestro dispositivo conectado sea estatica, ya que de no ser así tendríamos que actualizar la regla de redireccionamiento en nuestro router cada vez que esta IP privada cambie (veremos como hacer esto en el próximo apartado).

Teniendo esto en cuenta, si la IP privada de nuestro dispositivo es 192.168.1.104 tendriamos que crear la regla de redireccionamiento del puerto 80 a esta dirección IP de la siguiente manera:

No.	LAN IP	Tipo de Protocolo	Puerto en la LAN	Puerto público	Activar	Configurar
1	192.168.1.104	TCP&UDP	80	80	✓	

Figura 66: Redirección de puerto 80 a la dirección IP de nuestro dispositivo

La forma de crear la regla de redireccionamiento en el router depende del modelo de router en concreto con el que trabajemos. En nuestro caso hemos utilizado el router LiveBox de la empresa Orange y hemos accedido a esta opción mediante la ruta **Avanzada > NAT > Mapeo de Puertos**, pero como hemos dicho depende del modelo concreto del router.

Como podemos ver en la figura 64 el nombre del dominio asociado a nuestra *IP* pública es smartank.ddns.net, por tanto gracias a No-IP podremos acceder siempre a nuestro dispositivo a través de la URL:

```
http://smartank.ddns.net
```

Con esto terminamos de exponer todos los aspectos que hemos tenido que tener en cuenta a la hora de desarrollar una aplicación en Android y utilizarla como herramienta de gestión y visualización para nuestro dispositivo conectado.

7.2. Utilizando una *Plataforma IoT*

En el capítulo anterior vimos como crear un canal en *ThingSpeak* y cómo obtener las claves necesarias para conseguir comunicarnos con el canal. Además adaptamos el programa de nuestro dispositivo *NodeMCU* para establecer conexión con la Plataforma IoT y explicamos tanto la forma de construir la petición *HTTP* como la forma de enviar dicha petición a nuestro canal dentro de *ThingSpeak*, incluyendo el dato obtenido mediante el sensor que venimos utilizando.

En este apartado vamos a detenernos a analizar **cómo debemos configurar el canal de *ThingSpeak* asociado a nuestro proyecto**, de manera que tratemos los datos enviados por el dispositivo correctamente y podamos conocer el estado del depósito monitorizado en todo momento. Además también utilizaremos, de la misma forma que en la alternativa anterior, el servicio web *IFTTT* para notificar

por email al usuario en caso de ser necesario, gracias a las herramientas que *ThingSpeak* nos proporciona y que también estudiaremos en este capítulo.

7.2.1. Canal de ThingSpeak

Cuando creamos el canal vimos que se nos pedía introducir una serie de campos para su personalización. Vamos a ver ahora más detenidamente los más útiles de estos campos y aquellos que vamos a utilizar para nuestro proyecto:

- **Nombre**: simplemente el nombre que le asignaremos al canal para poder identificarlo facilmente. Sabemos que internamente *ThingSpeak* identifica a los canales por un número, que es único y generado automaticamente por la *Plataforma*, pero este nombre es más ilustrativo para nosotros.
- **Descripción**: un breve texto que podemos introducir para explicar la función de este canal.
- **Field 1 … Field 8**: Podemos marcar las casillas de tantos campos como necesitemos, y asignarles un nombre identificativo. Cada campo guardará los datos que nosotros le enviemos desde los dispositivos de que dispongamos.
- **Metadata**: En este apartado podemos incluir información relativa a los datos que contendrá nuestro canal, incluyendo datos en formato JSON, XML o CSV.
- **Etiquetas**: Podemos introducir palabras clave que servirán para identificar mejor nuestro canal.
- **Datos de localización**: Para conocer donde se encuentra físicamente nuestro dispositivo conectado podemos introducir manualmente su posición mediante las coordenadas de latitud, longitud y elevación. Por ejemplo si queremos indicar que nuestro dispositivo se encuentra en la ciudad de Jaén debemos introducir los siguientes datos:
    - **Latitud**: 37.77 º
    - **Longitud**: -3.78 º
    - **Elevación**: 491 m

- **Enlace a sitio externo**: Si tenemos un sitio web asociado al proyecto podemos indicarlo en este apartado.
- **Video URL**: Si tenemos un video alojado en las plataformas YouTube™ o Vimeo® que muestre información sobre nuestro canal podemos indicar el enlace a dicho video en este apartado.

Como podremos observar una vez tengamos creado nuestro canal, este se compone de una serie de apartados que vamos a pasar a ver a continuación.

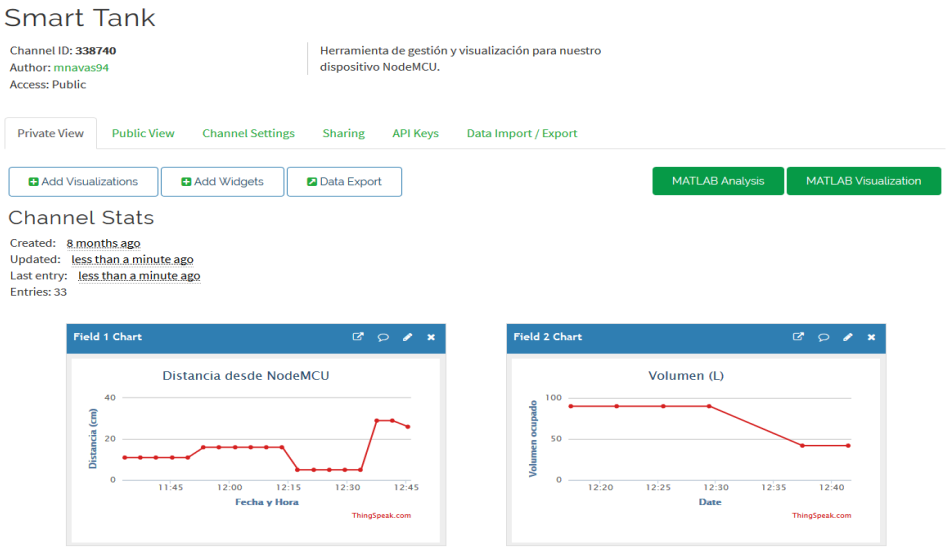


Figura 67: Pantalla principal del canal en ThingSpeak

En primer lugar en la parte superior se nos muestra un **resumen de la información sobre el canal**, indicandonos tanto el nombre que le hemos asignado nosotros como el identificador generado automaticamente por la Plataforma. Además del tipo de accesibilidad que tiene el canal (público o privado), la cuenta del propietario del canal y su descripción.

Después nos encontramos con una serie de pestañas donde se clasifican las distintas opciones que tenemos a la hora de utilizar y configurar el canal.

En la pestaña "**Channel Settings**" tendremos acceso a visualizar y modificar la configuración del canal que hemos introducido cuando lo hemos creado. Además se nos da la opción tanto de borrar todos los datos que contienen los campos (Fields) del canal, es decir de inicializarlo, como de eliminar completamente el canal si no vamos a volver a utilizarlo.

En la pestaña "**Sharing**" podremos controlar quien puede visualizar los datos contenidos en nuestro canal. Existen tres opciones diferentes para este apartado:

- Establecer el canal como **privado**, de manera que solamente el usuario de ThingSpeak que ha creado este canal tiene acceso a ver su contenido.
- Establecer el canal como **público**, para permitir de esta manera el acceso a la vista pública de nuestro canal.
- Establecer el canal como **público para los usuarios seleccionados**, es igual que la opción anterior pero esta vez solamente será accesible a aquellos usuarios de ThingSpeak con los que deseemos compartirla.

Dependiendo de la privacidad y el proposito de nuestra aplicación deberemos seleccionar la opción apropiada. Además se puede acceder a esta sección en cualquier momento y cambiar nuestras preferencias.

El siguiente apartado que vamos a analizar se trata de la pestaña **API Keys**. Como ya vimos en el apartado correspondiente del capítulo anterior, en este apartado tendremos acceso a las claves tanto de escritura como de lectura de nuestro canal y podremos generar nuevas claves en función de si necesitamos añadir más clientes de lectura o si creemos que nuestra clave de escritura (que es única) se ha visto comprometida y deseamos sustituirla por una nueva. Para más información sobre las claves de ThingSpeak y cómo se utilizan puede dirigirse al apartado 6.1.2 Comunicación con una *Plataforma IoT*.

Pasamos ahora a la pestaña **Data Import/Export** donde, como podemos intuir por su nombre, se nos ofrece tanto la opción de importar datos a nuestro canal directamente a partir de un archivo en formato CSV como de exportar todos los datos almacenados en el canal a un archivo CSV que se descargará a nuestro PC. Es una buena opción para realizar copias de seguridad de los datos de nuestro canal, para prevenir una posible corrupción de estos datos o la inserción en el canal de datos erroneos tanto por parte del cliente como por terceros que hayan conseguido obtener de forma ilícita la clave de escritura del canal. Por tanto es muy recomendable realizar periódicamente este tipo de respaldos para prevenir estos problemas.

Por último vamos a ver tanto el apartado **Private View** como el **Public View** ya que son similares. Tanto la vista pública como la privada nos ofrece información sobre los datos contenidos en los campos de nuestro canal. Esto se muestra normalmente a través de gráficas que nosotros podemos personalizar a nuestro gusto, aunque también pueden añadirse distintos Widgets para personalizar aun más las vistas de nuestro canal.

La principal diferencia entre ellas es que la vista privada solamente es accesible por el usuario del canal y la vista pública puede ser accesible (si nosotros lo establecemos así) para otros usuarios. Además podemos personalizar estas vistas por separado, de manera que nosotros tengamos acceso a más datos en la vista privada y en la vista pública se muestren únicamente los campos que nosotros deseemos.

Como hemos comentado, podemos crear una gráfica para cada uno de los campos que componen nuestro canal. Nosotros en nuestro caso hemos utilizado dos campos, uno para recolectar los datos de **distancia** que recoge el sensor, y otro distinto para almacenar los mismos datos pero transformados de forma que se muestre el **nivel del depósito** (en forma de porcentaje de volumen ocupado por el líquido) en función de las medidas y la forma de dicho depósito. En la vista privada mostraremos la información de ambos campos, mientras que en la vista pública solamente se podra acceder a la información sobre el nivel del depósito, pues es la información que realmente interesa al cliente que utilice esta herramienta para conocer el estado de su depósito monitorizado.

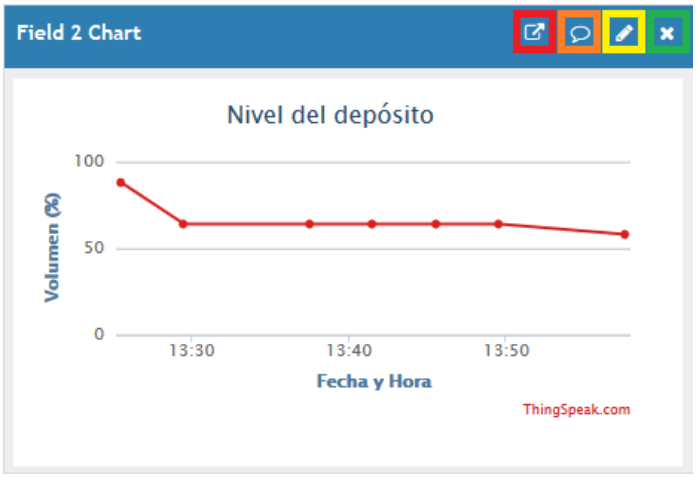


Figura 68: Gráfica del campo "Nivel del depósito" en ThingSpeak

Como se observa en la figura anterior, podemos realizar distintas acciones sobre estas gráficas haciendo clic en cada uno de los iconos remarcados en la imagen:

- El icono remarcado en **rojo** permite abrir una nueva pestaña en el navegador que contendrá solamente la gráfica en sí.
- El icono remarcado en **naranja** nos genera un código HTML con un objeto IFrame que nos permitirá insertar la gráfica en un documento web en formato HTML.
- El icono remarcado en **amarillo** nos da acceso al formulario de configuración de la gráfica, donde podremos modificar distintos aspectos que influirán en la forma en que se visualiza este gráfico. Entre otros campos podemos destacar:
    - **Titulo** del gráfico.
    - Nombre del **eje X**.
    - Nombre del **eje Y**.
    - **Color** de la gráfica.
    - Color de **fondo**.
    - **Tipo** de gráfico: linear, barras, columnas, etc.
    - Número máximo de **días** a visualizar.
    - Número máximo de **datos** a visualizar.
    - Valor **mínimo** del eje Y.
    - Valor **máximo** del eje Y.
- Por último el icono señalado en verde en la figura 68 elimina la gráfica de la vista donde nos encontremos (pública o privada).

Como hemos comentado anteriormente, podemos añadir en todo momento nuevas gráficas para representar cada uno de los campos de nuestro canal, además de distintos Widgets, como por ejemplo un termómetro o un medidor de velocidad.

Además de estas gráficas, las pestañas de vista pública y privada nos ofrece una serie de información de interes sobre el canal. En concreto hablamos del tiempo transcurrido desde la creación del canal, desde la última actualización, el último

envío de datos a un campo y el número de entradas o envíos que se han producido hasta el momento.

Por otra parte contamos con diferentes botones que nos permiten añadir, como ya hemos comentado, una nueva gráfica de visualización, un nuevo Widget o exportar los datos actuales. Además desde aquí podemos acceder a las herramientas de MATLAB disponibles en ThingSpeak, en concreto MATLAB Analysis y MATLAB Visualization. Estas forman parte de las herramientas que nos ofrece la Plataforma IoT ThingSpeak para mejorar la productividad de nuestros proyectos. Vamos a pasar a continuación a ver algunas de ellas, en concreto aquellas que nosotros hemos utilizado para nuestro proyecto.

7.2.2. Herramientas de ThingSpeak

Como ya sabemos estas aplicaciones son herramientas que nos ofrece esta Plataforma IoT para añadir más funcionalidades a nuestros proyectos. Con ellas podemos transformar los datos almacenados en el canal, representar de diferentes formas dichos datos y crear disparadores que ejecuten una acción determinada cuando se cumpla un criterio que nosotros mismos estableceremos.

Las más importantes de estas herramientas son las que funcionan bajo el software de MathWorks (MATLAB) alojado en la nube, que nos permiten programar y ejecutar código sin necesidad de adquirir una licencia de uso de MATLAB. Sin embargo esta característica presenta algunas limitaciones que debemos tener en cuenta:

- Limitaciones sobre la frecuencia y actualizaciones:
    - Podemos establecer “TimeControl”, es decir ejecutar nuestro código de MATLAB cada cierto periodo de tiempo que estableceremos para ejecutar tareas periódicamente (cada cierto número de minutos, horas, dias, etc). Sin embargo hay que tener en cuenta que el intervalo mínimo que podemos establecer para esta función es de 5 minutos, o lo que es lo mismo no podemos establecer la ejecución automática de un código en intervalos menores a 5 minutos.

  - Los puntos que se representarán en una gráfica creada mediante la herramienta MATLAB Visualizations añadida a un canal se actualizarán transcurridos 10 minutos.

- Limitaciones adicionales que producirán un error son:
  - Para los tipos de licencias “Standard” y “Academic” de ThingSpeak los códigos en las herramientas de MATLAB que requieran un tiempo de ejecución de más de 60 segundos en la nube producirán un error. Para los demás tipos de licencias, incluidas las cuentas gratuitas, los códigos que requieran más de 20 segundos de ejecución en la nube producirán error.
  - Para los tipos de licencia “Standard”, “Academic”, “Home” y “Student”, los códigos que escriban datos en un canal con una tasa inferior al segundo producirá un error. Para usuarios con cuenta gratuita, la misma limitación se amplía a tasas inferiores a los 15 segundos.
  - El código de la herramienta MATLAB Analysis que contenga funciones de visualización producirá un error siempre.

Como vemos son unas limitaciones bastante importantes asociadas sobre todo al tipo de licencia de ThingSpeak que hallamos adquirido. Para nosotros (recordamos que estamos utilizando una cuenta gratuita de ThingSpeak), sin embargo, estas limitaciones no suponen ningun impedimento para poder utilizar este tipo de herramientas para nuestro proyecto, pero sí es necesario conocerlas para prevenir problemas futuros.

Pasamos ahora a ver cada una de las herramientas que hemos utilizado en nuestro proyecto:

La primera herramienta es **MATLAB Analysis**, la cual nos permitirá analizar y manipular nuestros datos de diversas formas para después escribir nuevos datos en un canal de ThingSpeak o compartirlos con quien deseemos. Concretamente esta herramienta nos permitirá:

- **Explorar los datos** almacenados en un canal o los obtenidos desde un sitio web.

- **Encontrar y eliminar** datos corruptos.
- **Convertir** datos a diferentes unidades de medida.
- **Calcular** nuevos datos.
- **Construir modelos** de datos.

Además si disponemos de una cuenta de MathWorks® con la licencia apropiada podremos hacer uso de una serie de herramientas adicionales, tales como:

- **Statistics and Machine Learning Toolbox™**: Funciones para realizar estadisticas y sobre aprendizaje automático.
- **Image Processing Toolbox™**: Funciones de procesamiento de imágenes.
- **Text Analytics Toolbox™**: Funciones para análisis de textos.

Entre otras muchas que podemos encontrar en el sitio web de MathWorks. Para acceder a esta herramienta MATLAB Analysis debemos dirigirnos a la ruta **Apps > MATLAB Analysis** y crear una nueva instancia.

Tendremos que indicar un nombre para nuestro programa y escribir el código que se ejecutará en la nube de MathWorks, teniendo en cuenta las limitaciones que hemos comentado anteriormente. El código que hemos implementado nosotros es el siguiente:

```
1.  // ID del canal de lectura
2.  readChannelID = [338740];
3.  // APIKey de lectura del canal
4.  readAPIKey = '*********';
5.
6.  // ID del canal de escritura
7.  writeChannelID = [338740];
8.  // APIKey de escritura del canal
9.  writeAPIKey = '*********';
10.
11. // Leemos el dato de distancia medido por el sensor
12. [distance, time] = thingSpeakRead(readChannelID, 'Fields', 1);
13.
14. // Calculamos el nivel del depósito
15. totalVolume = (50 * 50 * 150)/1000; // Volumen total del depósito
16. occupiedVolume = (50 * 50 * (150 - distance))/1000; // Volumen ocupado
17. percent = (occupiedVolume * 100)/totalVolume; // Porcentaje del nivel del depósito
18.
19. // Escribimos el dato del nivel del deposito en el campo correspondiente
20. thingSpeakWrite(writeChannelID, percent, 'Fields', 2,
                    'TimeStamps', time, 'WriteKey', writeAPIKey);
```

En este caso, como podemos ver en el código anterior, hemos decidido implementar el código para monitorizar un depósito de tipo rectangular y con unas dimensiones de 50 cm x 50 cm x 150 cm. Si queremos modificar el tipo de depósito o sus dimensiones tendríamos que acceder al apartado correspondiente dentro de la Plataforma IoT y cambiar el código del objeto MATLAB Analytics (o crear otro objeto distinto para otro tipo de depósito). Si hacemos clic en "Save and Run" podemos ejecutar nuestro código y comprobar que funciona correctamente. En nuestro caso este código coge el último dato almacenado en el campo "Distancia" y lo utiliza para calcular el nivel del depósito en relación al volumen total del depósito, después este dato se guarda en el mismo canal pero en el campo 2 "Volumen ocupado" que se corresponde con la gráfica que hemos configurado en la vista pública del canal.

Otra de las herramientas que nos ha permitido utilizar esta Plataforma IoT como herramienta de visualización para nuestro proyecto es **TimeControl**. Está relacionada con la herramienta anterior, pues se encarga de ejecutar el código que hemos escrito en el objeto MATLAB Analytics periódicamente según nuestras preferencias.

Para añadirlo debemos dirigirnos a la ruta **Apps > TimeControl** y crear una nueva instancia. Nos mostrará un formulario para seleccionar la frecuencia de ejecución (nosotros hemos seleccionado el mínimo que es 5 minutos) y la acción que queremos desarrollar, en nuestro caso el código de MATLAB Analysis.

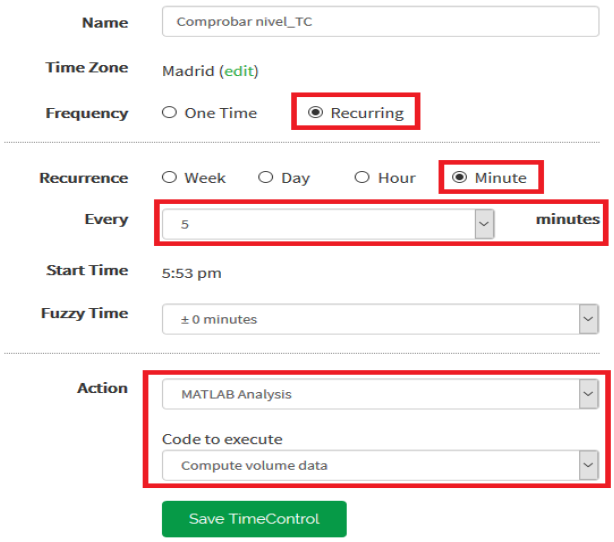


Figura 69: Configuración de TimeControl en ThingSpeak

Por último vamos a utilizar otra herramienta parecida a TimeControl que nos permitirá enviar un email que notificará al usuario sobre el nivel del depósito. Para ello necesitaremos la herramienta **ThingHTTP**. Esta herramienta permite la comunicación entre dispositivos, servicios web y sitios web mediante peticiones HTTP. De esta manera comunicaremos al servicio web que utilizamos en la alternativa anterior (IFTTT) de la misma forma (petición *HTTP* mediante el método *POST* tal y como explicamos en el apartado 7.1.4 *Comunicación con el dispositivo* conectado) para enviar un correo electrónico a una dirección determinada en caso de cumplirse la condición.

Para esta tarea debemos crear una instancia de esta herramienta accediendo a la ruta **Apps > ThingHTTP** dentro del sitio web de *ThingSpeak*. Al igual que pasaba con la herramienta anterior debemos rellenar el formulario que generará nuestro objeto *ThingHTTP*. Entre otros campos, los más importantes que debemos completar son:

- **Nombre** del objeto que estamos creando,
- **URL** donde enviaremos la petición, que en nuestro caso es:

```
1.  https://maker.ifttt.com/trigger/notificar_email/with/key/*******
```

  Donde notificar_email es el **nombre del evento de IFTTT** que ya vimos y "*******" es la **clave privada de IFTTT**.

- Tipo del método HTTP que vamos a utilizar. En nuestro caso es el método **POST**.
- **Tipo de contenido**: Debemos indicar que estamos enviando un texto en formato JSON, por tanto en este campo debemos incluir el texto "application/json" para que IFTTT reconozca nuestros datos correctamente.
- **Versión de HTTP**: 1.1.
- **Cuerpo** de la petición HTTP, que lo rellenaremos con la información que queremos enviar al servicio de IFTTT, tal y como ya hemos visto anteriormente:

```
1.  {"value1" : "smartank2018@gmail.com", "value2" : "20"}
```

Ya tenemos configurado nuestra instancia de *ThingHTTP* que enviará la petición a *IFTTT*, pero necesitamos implementar una herramienta más que controle cuando se ejecutará esta petición. Para esto necesitamos implementar una instancia de la herramienta **React** en la ruta **Apps > React**. Esta herramienta tan solo se ocupará de **comprobar cada cierto tiempo si se cumple la condición** que nosotros le indiquemos con los datos de nuestro canal y, en caso afirmativo, **generará la acción** que también le determinaremos nosotros.

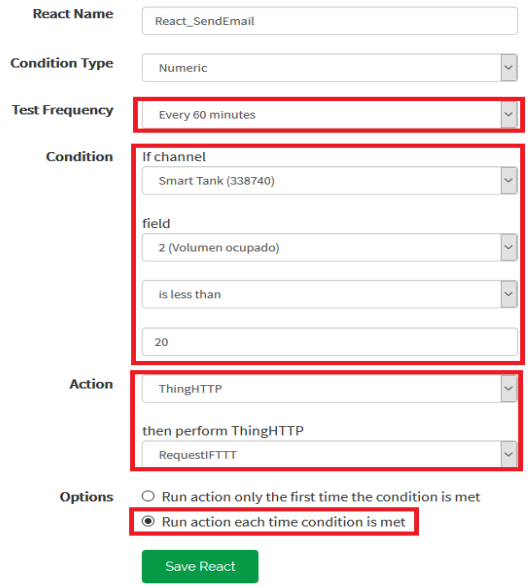


Figura 70: Configuración de React en ThingSpeak

Como vemos en la figura anterior hemos establecido que cada 60 minutos se compruebe si el valor del campo 2 (el que contiene el nivel del depósito en porcentaje) de nuestro canal disminuye por debajo del valor 20 (que se corresponde con el 20% del nivel del depósito) para posteriormente enviar la petición *HTTP* al Servicio Web *IFTTT* mediante el objeto *ThingHTTP* que hemos creado anteriormente. Además establecemos que se produzca esta acción cada vez que la condición establecida se cumpla para que se ejecute siempre que el nivel baje del umbral establecido.

Además de las herramientas que hemos explicado podemos encontrar algunas otras que pueden ser de utilidad en cualquier proyecto relacionado con el IoT. Para conocer todas las herramientas disponibles en *ThingSpeak* puede dirigirse a la sección de ésta *Plataforma IoT* dentro del capítulo 3.2.

7.2.3. ThingView

Para facilitar la **visualización del estado del depósito monitorizado desde un dispositivo móvil** existe una aplicación para smartphones que permite ver la vista pública de nuestro canal fácilmente y sin necesidad de recordar URLs complejas. Esta aplicación se encuentra disponible **tanto para dispositivos con SO Android como para aquellos que utilicen iOS** como sistema operativo, lo cual hace muy accesible poder utilizar esta aplicación pues son los sistemas más frecuentes en el mercado actual de telefonos inteligentes.

Esta aplicación se trata de **ThingView** y es totalmente gratuita. Tan solo tendremos que acudir a la tienda de aplicaciones de nuestro dispositivo (*Google Play* o *AppStore*) y descargarla. Además es una aplicación muy fácil de usar, como se puede observar en al figura 71, ya que solamente deberemos introducir el **número de identificación único de nuestro canal** de *ThingSpeak* y confirmar que se trata del canal correcto. Después podremos seleccionar la gráfica que queremos visualizar, en caso de que dispongamos de más de una, y automaticamente se nos mostrará en pantalla como si estuvieramos accediendo al sitio web de *ThingSpeak*.

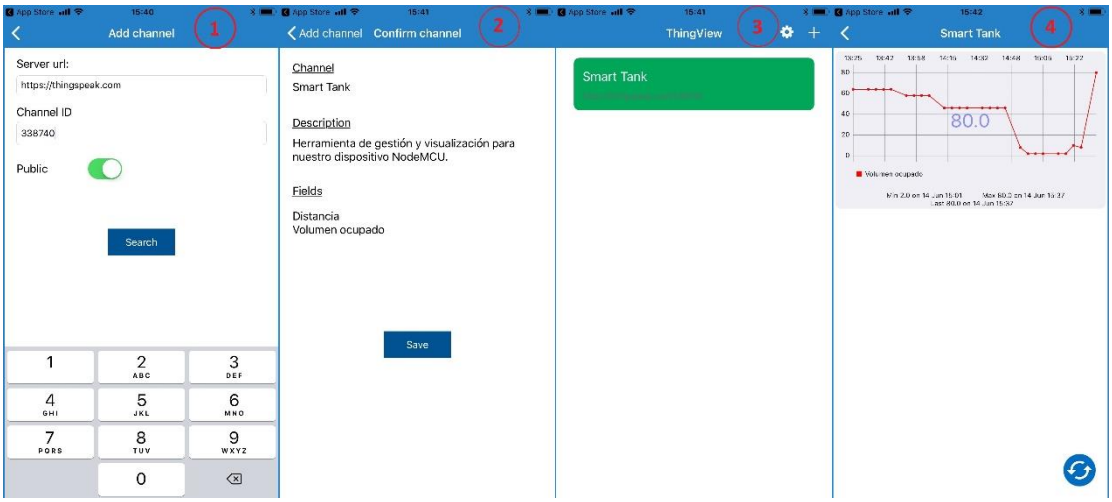


Figura 71: Configuración de nuestro canal de ThingSpeak en ThingView

Con esto terminamos de explicar los aspectos más importantes que hemos tenido que estudiar y solucionar para poder desarrollar la aplicación de gestión y visualización para nuestro producto IoT utilizando una Plataforma IoT.

Capítulo VIII

Conclusiones

Para concluir este trabajo vamos a hacer un breve recopilatorio de todo lo que hemos visto a lo largo del mismo y terminaremos con unas impresiones finales del proyecto y el tema que hemos tratado.

En este trabajo hemos introducido el concepto de **Internet de las Cosas** (*IoT*), comenzando por los antecedentes, como el nacimiento y la expansión de Internet, para después embarcarnos en el propio concepto en sí y todas las **tecnologías** y elementos relacionados, como son las placas de desarrollo, las tecnologías de comunicación, el *Cloud Computing* y la analítica de datos. Hemos comprobado que es una tecnología que ha progresado enormemente durante los últimos años conforme los avances tecnológicos lo han permitido, y también hemos demostrado que cuenta con un **gran futuro**, con grandes posibilidades de ampliación y de generar **nuevos modelos de negocio** para una serie de industrias emergentes (*SmartHome*, *SmartCity*, salud, etc.) que se encuentran actualmente dando sus primeros pasos, y que en un futuro no muy lejano seguro que van a tener un gran mercado que generará muchos beneficios.

Se han introducido también unas herramientas que se han desarrollado a lo largo de los últimos años de la mano del concepto de Internet de las Cosas: las **Plataformas IoT**. Hemos visto las **ventajas** que supone su utilización en cualquier proyecto relacionado con el Internet de las Cosas, y hemos **comparado** distintos tipos de *Plataformas IoT* para desentrañar sus características, sus puntos fuertes, las desventajas y los tipos de proyectos donde se utilizan. Para terminar **valoramos** cual era la mejor *Plataforma* de las

analizadas para emplear en nuestro proyecto, teniendo en cuenta las características y requisitos del mismo, siendo finalmente **ThingSpeak** la seleccionada.

Este fue el primer paso en nuestro proyecto, donde el propósito final era desarrollar un prototipo de producto IoT que nos permitiera monitorizar un depósito de manera que podamos conocer en cualquier momento su estado y vigilar que su nivel no disminuya por debajo de un umbral establecido. Sin embargo este proposito no es más que el camino para el verdadero objetivo que hemos perseguido durante este trabajo, que no es otro que dar una **visión general de todo lo que implica el desarrollo de un producto IoT completo**, desde el hardware necesario para llevar a cabo nuestro propósito, pasando por la programación del MCU, los sensores que necesitabamos y diversos aspectos relacionados (almacenamiento en memoria no volátil, conexión a la red WiFi, etc) hasta la implementación de una herramienta de gestion y visualización a través de dos formas totalmente distintas: utilizando una Plataforma IoT y desarrollando una aplicación a medida para dispositivos móviles con sistema operativo Android.

Esto último nos ha permitido **comprobar de primera mano** la facilidad y rapidez en cuanto a desarrollo que supone utilizar una Plataforma IoT en comparación con otras alternativas menos especializadas. Aunque también hemos comprobado que si queremos personalizar el 100% del proyecto según nuestras preferencias siempre será mejor realizar un desarrollo desde cero de una aplicación del tipo que sea (móvil, web, etc) a expensas de destinar más costes al desarrollo.

Como ya se ha comentado nuestro proyecto es simplemente un prototipo, y por tanto **hay muchas cosas que podríamos mejorar** en varios aspectos, como incluir una gestión de usuarios, establecer más tipos de depósitos con los que poder trabajar, mejorar la seguridad de las comunicaciones mediante el uso de certificados SSL, etc. Con el producto que hemos implementado hemos conseguido el objetivo principal de nuestro proyecto, dejando como **trabajo futuro** añadir mayor funcionalidad y mejorar nuestro desarrollo para conseguir un producto aún más complejo y perfeccionado.

Referencias

A. Al-Fuqaha, M. G. (2015). Internet of Things: A Survey on Enabling Technologies, Protocols and Applications. *IEEE Communications Surveys & Tutorials, 17*(4).

A. McEwen, H. C. (2014). Internet de las Cosas. La tecnología revolucionaria que todo lo conecta. *Anaya Multimedia*.

Aguilar, L. J. (2015). *Programación en C/C++, Java y UML.* McGraw-Hill Interamericana de España S.L.

Arduino Libraries. (2018). Obtenido de https://www.arduino.cc/en/Reference/Libraries

Arduino. (s.f.). *Official Arduino Reference*. Obtenido de https://www.arduino.cc/reference/

Butler, B. (2017). What is edge computing and how it's changing the network. *Network World*.

C. Perera, A. Z. (2014). Context Aware Computing for The Internet of Things: A Survey. *IEEE Communications Surveys & Tutorials*.

Chun-Wei Tsai, C.-F. L.-C. (2014). Data Mining for Internet of Things: A Survey. *IEEE Communications Surveys & Tutorials, 16*(1).

Cisco. (2015). Fog Computing and the Internet of Things: Extend.

Espressif Inc. (2017). *ESP8266 Technical Reference.*

Espressif Inc. (2018). *ESP32 Technical Reference.*

Evans, D. (2011). *Internet of Things.* IBSG de Cisco.

Fundación Princesa de Asturias. (2002). Obtenido de http://www.fpa.es/es/

Gartner. (Agosto de 2015). Obtenido de https://www.gartner.com/newsroom/id/3114217

Gartner. (Julio de 2017). Obtenido de https://www.gartner.com/newsroom/id/3784363

Google. (2018). *Google Trends*. Obtenido de https://trends.google.es/trends/

Hart, D. (2003). *National Science Fundation.* Obtenido de https://www.nsf.gov/news/news_summ.jsp?cntn_id=103050

IoT Platforms: Market Report 2015-2021. (2016). Obtenido de IoT Analytics: https://iot-analytics.com/

James F. Kurose, K. W. (2010). *Redes de computadoras: Un enfoque descendente.* Pearson Educación S. A.

M. A. Razzaque, M. M.-J. (2016). Middleware for Internet of Things: A Survey. *IEEE Internet of Things Journal, 3*(1).

M. Damas, F. G. (2017). La asignatura “Internet de las Cosas” en el master DATCOM de la UGR. *Enseñanza y aprendizaje de ingeniería de computadores: Revista de Experiencias Docentes en Ingeniería de Computadores*(7).

O. Vermesan, P. F. (2013). *Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems.* River Publishers.

R. Ciobanu, C. D. (2014). Big Data Platforms for the Internet of Things. *Big Data and Internet of Things: A Roadmap for Smart Environments, 3*(1).

Sandhya Chalasani, J. M. (2008). A survey of energy harvesting sources for embedded systems. *IEEE SoutheastCon*.

Schneider, S. (9 de October de 2013). Understanding the Protocols Behind The Internet Of Things. *Electronic Design*.

Shanhe Yi, C. L. (2015). A Survey of Fog Computing: Concepts, Applications and Issues.

SiLabs. (s.f.). *CP210x USB to UART Bridge VCP Drivers*. Obtenido de https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers

Vazhnov, A. (2015). *La Red de Todo: Internet de las Cosas y el Futuro de la Economía Conectada.*

W. Colitti, N. T. (2014). Embedded Web Technologies for the Internet of Things. *Internet of Things Challenges and Opportunities*.

W3C. (2001). *REST Semantic Web Standards*. Obtenido de https://www.w3.org/2001/sw/wiki/REST

W3C. (2018). *SOAP Specifications*. Obtenido de https://www.w3.org/TR/soap/

Weisong Shi, J. C. (Octubre de 2016). Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal, 3*(5).

Wikipedia. (2018). Obtenido de https://es.wikipedia.org/

Wikipedia. (2018). *Ciclo de sobreexpectación*. Obtenido de https://es.wikipedia.org/wiki/Ciclo_de_sobreexpectación

Anexos

A. Lenguaje Unificado de Modelado (*UML*)

Los diagramas de clase se componen principalmente de:

- **Clase**: Unidad básica de la lógica de nuestra aplicación. Encapsula información relativa a un tipo de objeto, contiene atributos, métodos y adquiere una visibilidad concreta. En los diagramas UML se representan por un rectángulo con tres divisiones: nombre de la clase, atributos y métodos**.**

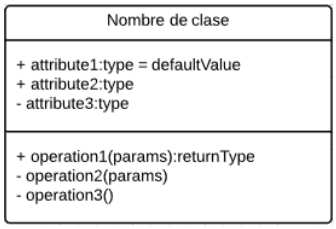


Figura 72: Representación de una clase en *UML*

- **Relaciones**: Indican cómo se comunican los objetos de las distintas clases entre sí. Existen varios tipos distintos de relaciones:

    - **Asociación**: relación estructural que describe una conexión entre objetos. Se representa con un línea continua que une las clases relacionadas entre sí. Suele ser una relación bidireccional pero es posible representar relaciones unidireccionales, en tal caso se indican con una flecha que indica el sentido de la relación. Además las asociaciones pueden tener multiplicidad que indican el número de objetos de cada clase que intervienen en la relación.

Multiplicidad	Significado
1	Sólo uno
0..1	Cero o uno
*****	Cero o varios
0..*	Cero o varios
1..*	Uno o varios (mínimo uno)
N..M	Desde N hasta M

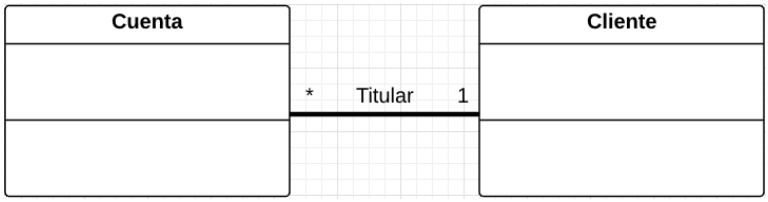


Figura 73: Ejemplo de una asociación en *UML*

- **Agregación**: Es un caso particular de una asociación. Representa una relación entre un todo y sus partes en la que las partes pueden formar parte de distintos agregados.

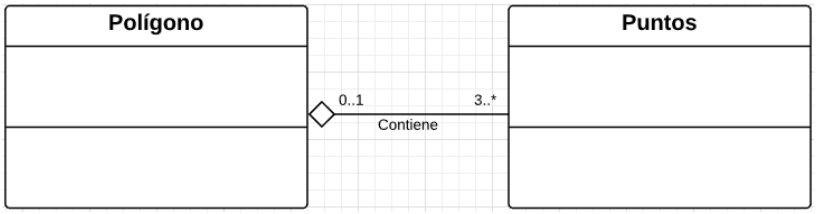


Figura 74: Ejemplo de una agregación en *UML*

- **Composición**: Parecida a la agregación, pero en esta relación las partes sólo existen asociadas al compuesto, es decir solo se pueden acceder a ellas a traves del compuesto.

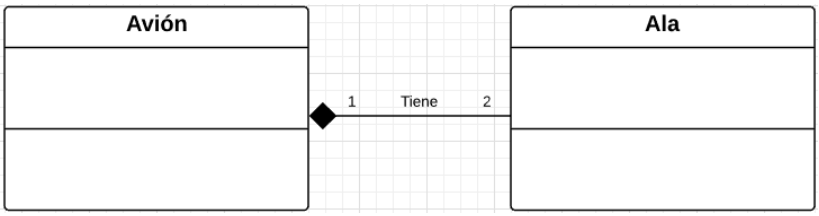


Figura 75: Ejemplo de una composición en *UML*

- **Dependencia**: Tipo de relación entre clases más débil que la asociación, indica la relación en la cual una clase hace uso de otra, bien instanciándola o recibiéndola como parámetro de entrada en alguno de sus métodos. Se representa con una línea discontinua con una flecha en el extremo que indica la clase que es utilizada.

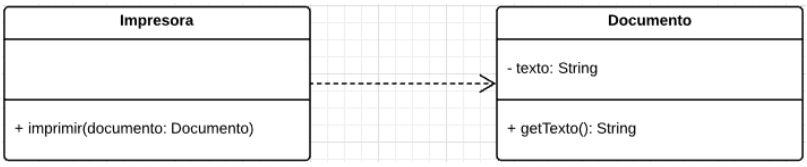


Figura 76: Ejemplo de una dependencia en *UML*

- **Herencia**: Indica la relación entre una superclase y sus subclases en la que todas las subclases heredan los métodos y atributos especificados en la superclase, por tanto cada subclase poseerá, además de sus propios métodos y atributos, todos aquellos de la superclase que sean visibles, es decir los declarados como *public* y *protected*.

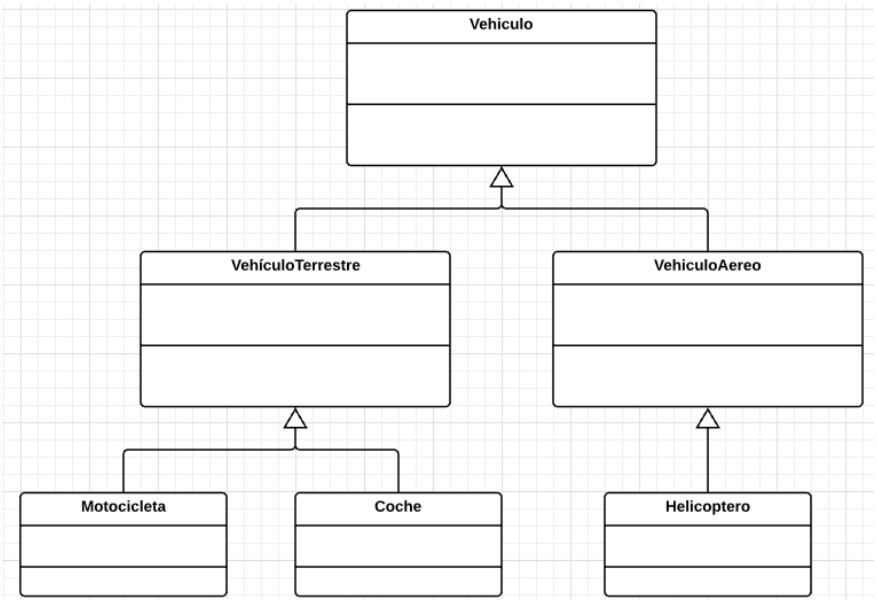


Figura 77: Ejemplo de herencia en *UML*

B. Instalación y configuración de *Android Studio*

Android Studio es el entorno de desarrollo donde crearemos nuestra aplicación, pero antes de empezar a sacarle partido tenemos que instalarlo correctamente en nuestro ordenador. Es importante asegurarnos antes de proceder a descargar e instalar el IDE que contamos con un equipo que cumpla los requisitos mínimos para ejecutarlo. Estos requisitos dependen del sistema operativo que se ejecute sobre nuestro hardware:

- Los requisitos para equipos con **SO Windows** son:
    - SO Windows 7/8/10, sin importar la arquitectura (32 o 64 bits)
    - 4 GB de memoria RAM (se recomiendan 8 GB)
    - Java Development Kit (JDK) 8 instalado
    - Pantalla con resolución de, al menos, 1280 x 800 píxeles
    - Mínimo 2 GB libres de almacenamiento en disco
- Los requisitos para equipos con **SO MacOS** son:
    - SO MacOS X 10.10 (Yosemite) o superior
    - 4 GB de memoria RAM (se recomiendan 8 GB)
    - Java Development Kit (JDK) 8 instalado
    - Pantalla con resolución de, al menos, 1280 x 800 píxeles
    - Mínimo 2 GB libres de almacenamiento en disco

Una vez que hemos comprobado que nuestro equipo cumple los requisitos de hardware podemos pasar al siguiente paso, instalar el JDK. El **JDK** o **Java Development Kit** no es más que un conjunto de herramientas que permiten comenzar a desarrollar en lenguaje Java. Obviamente si ya tenemos instalada la versión de Java 8 o superior no necesitamos realizar esta instalación. Para comprobar la versión del JDK instalada en nuestro equipo podemos abrir la ventana de comandos y escribir:

```
java --version
```

Lo que nos mostrará un mensaje similar al siguiente, donde podremos comprobar que versión tenemos instalada:

```
java 10 2018-03-20

Java(TM) SE Runtime Environment 18.3 (build 10+46)

Java HotSpot(TM) 64-Bit Server VM 18.3 (build 10+46, mixed mode)
```

En caso de no contar con la versión requerida del JDK tan solo tendremos que acudir a la web oficial de Java SE Development Kit y proceder a la descarga e instalación manteniendo las opciones predeterminadas.

Es importante recalcar que en la web de Android Studio se nos indica que en algunos sistemas de Windows, la secuencia de comandos de inicio no encuentra el destino de instalación del JDK. Si se produce este problema, debemos configurar una variable de entorno que indique la ubicación correcta. Para ello debemos seleccionar **Panel de control > Sistema y Seguridad > Sistema > Propiedades avanzadas del sistema**. Luego abrimos la pestaña **Opciones avanzadas > Variables de entorno…** y añadiremos una nueva variable de sistema con nombre **JAVA_HOME** que apunte a la carpeta donde hemos instalado el JDK. Si no hemos cambiado la ruta por defecto en la instalación del JDK debería ser una ruta similar a la siguiente:

```
C:\Program Files\Java\jdk1.8.0_77
```

Después de haber instalado y configurado correctamente el JDK podemos proceder a la instalación del IDE Android Studio. Para ello debemos acudir a la pagina web oficial de Android Studio y descargar la última versión disponible para nuestro sistema operativo, en nuestro caso es la 3.1.2 para Windows. Tras la descarga solamente tendremos que ejecutar el instalador y mantener las opciones por defecto.

Una vez instalado el entorno de desarrollo ya podemos ejecutarlo. Al ser la primera vez que abrimos el IDE tendremos que completar una pequeña configuración inicial donde se nos dará a elegir sobre varios aspectos importantes, concretamente:

- Se nos pedirá importar la configuración de una instalación anterior; seleccionamos “no importar configuraciones”.
- Se nos preguntará sobre el tipo de instalación que queremos; seleccionamos el tipo “estandar”.
- Se nos dará a elegir el tipo de interfaz que más nos agrade; podemos elegir cualquiera de las dos, nosotros seleccionamos la interfaz “IntelliJ”

Una vez concluida la instalación se nos mostrará el menú de bienvenida de *Android Studio*, donde podemos comenzar a trabajar con el *IDE*.

C. Instalación y configuración de *Arduino IDE*

Lo primero que debemos hacer es descargar el IDE de Arduino desde su página web oficial (www.arduino.cc). En el apartado “Software” encontraremos la sección del IDE de Arduino. Una vez descargado podemos proceder a su instalación manteniendo las opciones predeterminadas.

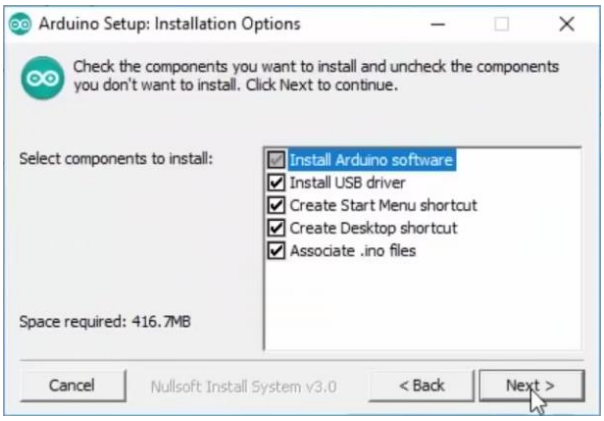


Figura 78: Instalación del IDE Arduino

Una vez finalizada la instalación procederemos a abrir el programa por primera vez y a su correcta configuración con ESP8266. Es importante para los pasos siguientes que añadamos una regla en nuestro Firewall para permitir el acceso a internet del IDE. Si usamos el Firewall de Windows se nos dará esta opción la primera vez que ejecutemos el IDE, pero si no deberemos hacerlo manualmente. Ahora procedemos con la configuración:

Hay que mencionar que esta configuración es la indicada si instalamos el IDE en un sistema operativo Windows. Si se desea instalar bajo un SO MacOS debemos realizar un paso adicional que explicaremos al final del proceso.

Opcional: El IDE por defecto utilizará el idioma del sistema, si queremos cambiarlo podemos hacerlo dentro de la pestaña “**File > Preferences**” y configurarlo en Español si no nos aparece en nuestro idioma. Para aplicar los cambios hay que reiniciar el programa.

Comenzaremos seleccionando el tipo de placa que vamos a programar dentro del menú “Herramientas”. Sin embargo si observamos la lista veremos que solo

aparecen las tarjetas con soporte oficial de arduino. Para añadir otras tendremos que abrir el Gestor de tarjetas, dentro de "**Herramientas > Placa**".

Escribimos **ESP8266** en el campo de búsqueda, y comprobaremos que no aparece ninguna tarjeta con ese nombre, esto ocurre porque debemos indicarle al IDE Arduino de donde tiene que descargar los archivos necesarios. Para ello debemos volver al navegador web y entrar en el repositorio oficial en github del framework para Arduino del ESP8266 www.github.com/esp8266/arduino donde encontraremos las instrucciones para instalar el soporte de esta tarjeta en el entorno de desarrollo de Arduino. Lo que vamos a necesitar es el enlace al archivo JSON en la descripción del proyecto.

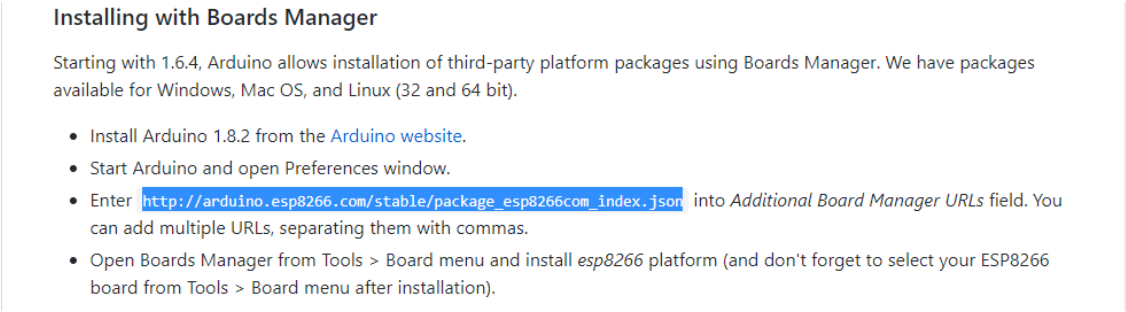


Figura 79: Repositorio en GitHub del framework para Arduino del ESP8266

Este archivo le describe al IDE los componentes que tiene que descargar para incorporar las placas ESP8266. Copiamos este enlace y volvemos al IDE.

Una vez de nuevo en el IDE entramos dentro del menú "**Archivo > Preferencias**" e ingresamos el enlace que hemos obtenido en el repositorio en el campo **Gestor de URLs adicionales de tarjetas**.

Figura 80: Gestor de URLs Adicionales de Tarjetas

Volvemos a buscar **ESP8266** en el campo de búsqueda del **Gestor de tarjetas**, dentro de "**Herramientas > Placa**", y ahora si nos aparecerá la opción para instalar los archivos que necesitamos.

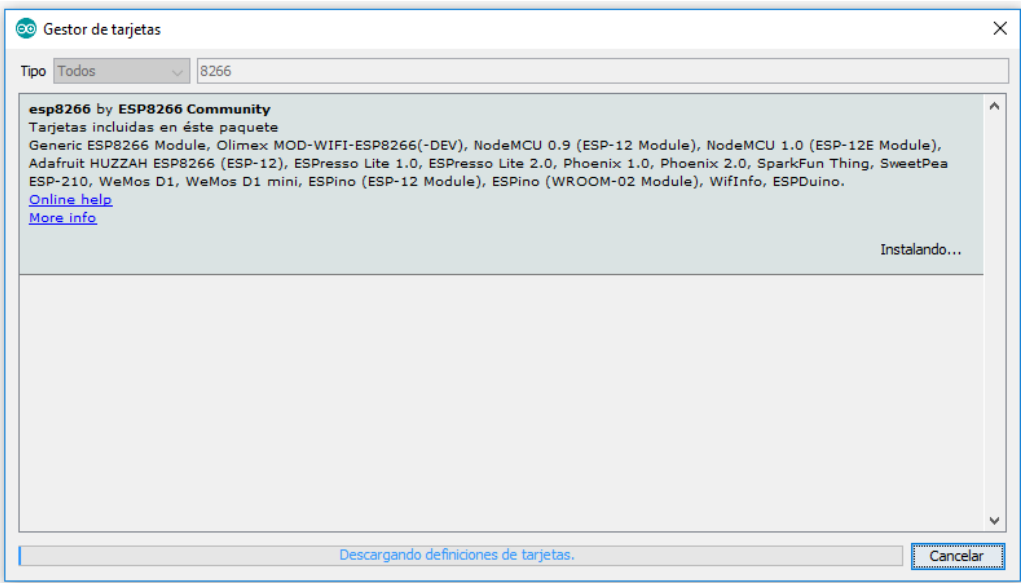


Figura 81: Gestor de tarjetas IDE Arduino

Volvemos a abrir el desplegable "**Herramientas > Placa**" y podemos observar que aparecen las tarjetas que incorporan el ESP8266. Seleccionamos **NodeMCU 1.0 (ESP-12E Module)** en nuestro caso.

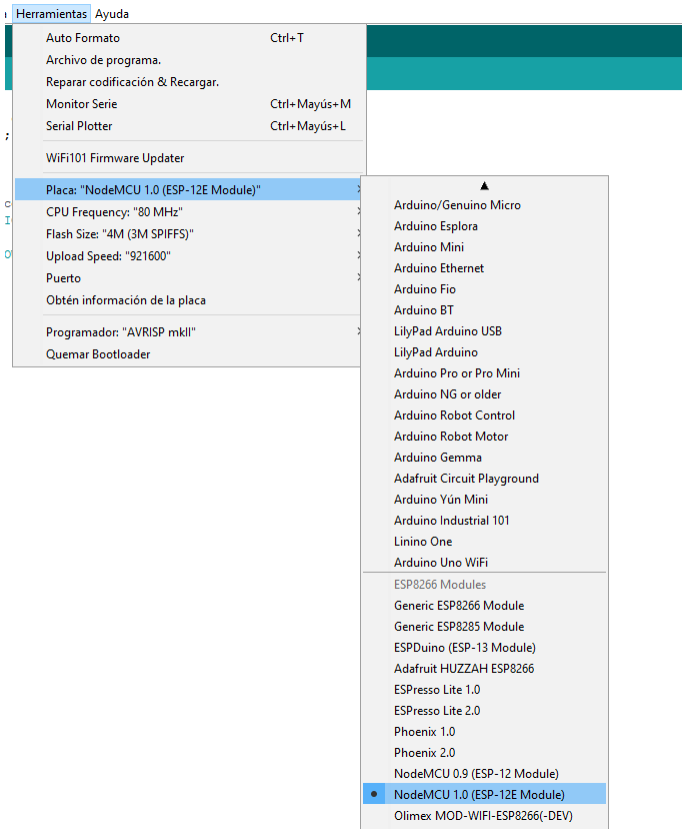


Figura 82: La versión de nuestra placa es NodeMCU 1.0 (ESP-12E Module)

Para configurar nuestra tarjeta debemos conocer un par de características sobre ella. Primero tenemos que conocer la **capacidad de la memoria flash**, dependiendo del modulo con el que vayamos a trabajar dicha capacidad puede variar, por ejemplo, el ESP-01 tiene una capacidad de 512 kB y el ESP-12 (nuestro caso) tiene 4 MB. Seleccionamos dicha capacidad en "**Herramientas > Flash size**" También tenemos que configurar la **velocidad del puerto serie** durante la carga del programa, cuanto mas alta sea menos tardara la carga del programa al microcontrolador. Para saber que valor es el adecuado en nuestro caso concreto no hay mas remedio que utilizar el método de prueba y error, ya que esta velocidad va a depender de la placa que estemos usando y de la calidad del cable USB. Inicialmente configuraremos la velocidad mas alta posible y si falla seleccionamos el siguiente valor más bajo de la lista, hasta que nos cargue el programa correctamente en el microcontrolador.

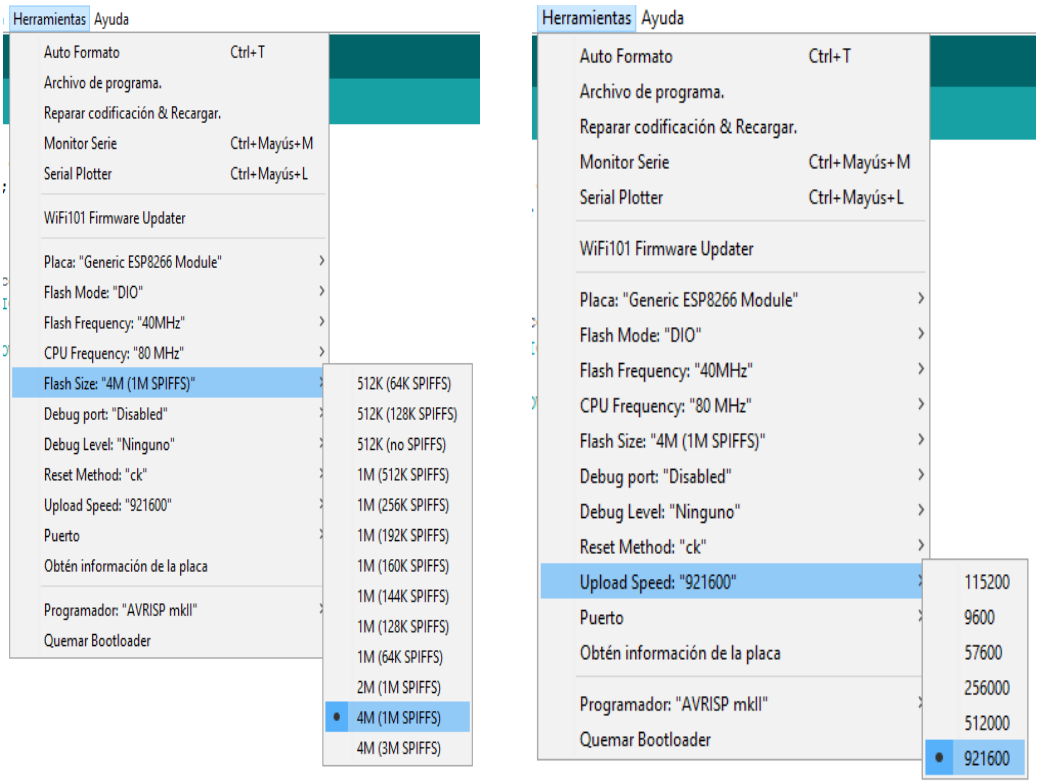


Figura 83: Configuración de la capacidad de memoria Flash y velocidad del puerto serie

Ahora ya podemos conectar nuestra tarjeta al PC con el cable USB. Una vez conectada, nos aparecerá en "**Herramientas > Puerto**" el puerto serie que corresponde con el conversor de USB a RS-232. En nuestro caso es el puerto COM 3.

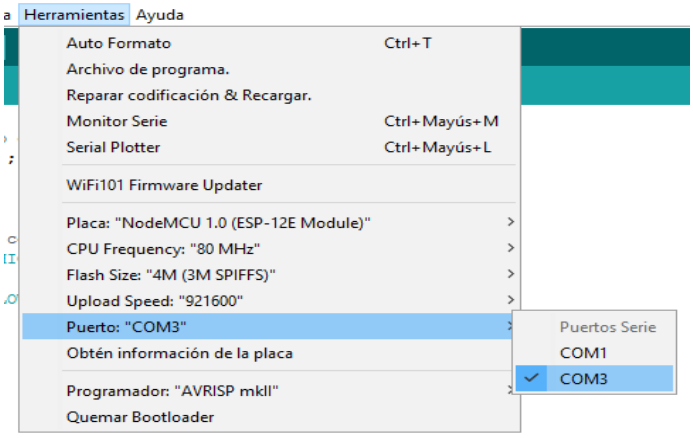


Figura 84: El ultimo paso es seleccionar el puerto correspondiente a nuestra placa

Ya tenemos todo configurado para cargar nuestros programas en nuestra placa NodeMCU.

En caso de querer configurar el IDE para programar nuestra placa desde un sistema operativo MacOS debemos realizar un paso adicional. Debemos instalar los drivers de **SiLabs** para crear un puente virtual de USB a puerto COM (VCP, virtual COM port), lo cual es necesario para conseguir la comunicación entre nuestro dispositivo y el ordenador a traves del puerto USB de este. Para descargar estos drivers debemos dirigirnos al sitio web de SiLabs (SiLabs, s.f.)

Una vez descargado, en el menú de selección del puerto donde tenemos conectada nuestra tarjeta, seleccionaremos la opción "**cu.SLAB.USBtoUART ”**.

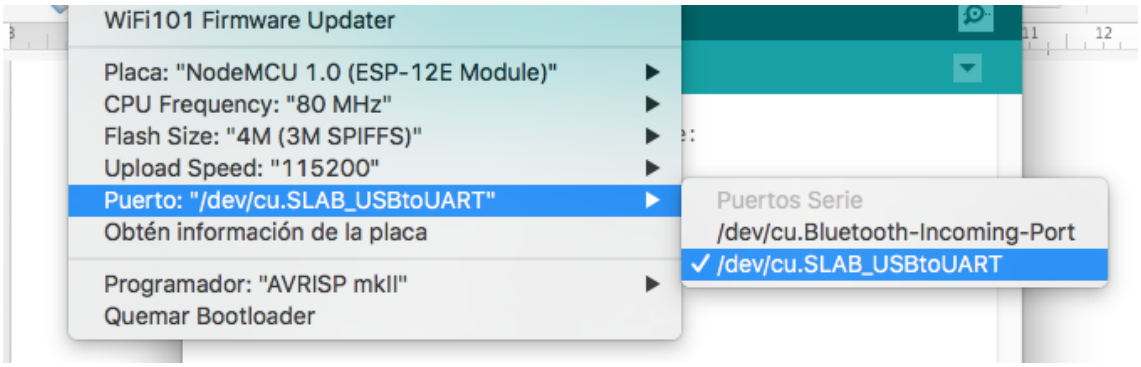


Figura 85: Configuración drivers SiLabs para MacOS

Si queremos comprobar que la instalación y configuración se ha realizado satisfactoriamente podemos cargar alguno de los programas de ejemplo que encontramos en el IDE. Para hacerlo solamente debemos dirigirnos a "**Archivo > Ejemplos**" y seleccionar alguno de ellos. Después podremos cargarlo en nuestra placa mediante el botón de "**Subir**" que se encuentra en la parte superior izquierda representado por una flecha hacia la derecha.