UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Caronte.io: Una aplicación para la ayuda a la citación mediante dispositivos móviles

Trabajo Fin de Máster

Máster Universitario en Ingeniería Informática

Autor: César Martín Fort Palau

Tutores: Vicente Pelechano Ferragud

Andrés Boza García

2016-2017

Resumen

Este trabajo de fin de máster pretende diseñar e implementar un sistema de información destinado a mejorar la llegada y recepción de los pacientes en una clínica. El objetivo es mejorar la calidad asistencial y la satisfacción de los pacientes facilitándoles las indicaciones necesarias para llegar a su cita y reducir los tiempos de espera mediante la utilización de técnicas de inteligencia ambiental e Internet of things.

El proyecto se implementará mediante el desarrollo de una aplicación móvil Android y un módulo complementario que permita integrar el software desarrollado con las aplicaciones clínicas existentes. El sistema se podrá adaptar fácilmente a otros ámbitos de aplicación donde el desplazamiento a una cita sea un factor susceptible de mejora: reuniones de trabajo, servicios públicos, tutorías académicas, etc.

Palabras clave: Internet of things, informática médica, gestión hospitalaria, citación, Android.

Abstract

This master's degree project aims to design and implement an information system designed to improve the arrival and reception of patients in a clinic. The objective is to improve the quality of care and patient satisfaction by providing them with the necessary information to reach their appointment and reduce waiting times through the use of environmental intelligence and Internet of things techniques.

The project will be implemented through the development of an Android mobile application and a complementary module that allows the integration of the developed software with the existing clinical applications. The system will be able to adapt easily to other fields of application where going to an appointment is a factor susceptible of Improvement: work meetings, public services, academic tutorials, etc.

Keywords: Internet of things, medical computing, hospital management, appointment scheduling, Android.

Resum

Aquest treball de fi de màster pretén dissenyar i implementar un sistema d'informació destinat a millorar l'arribada i recepció dels pacients en una clínica. L'objectiu és millorar la qualitat assistencial i la satisfacció dels pacients facilitant-los les indicacions necessàries per a arribar a la seua cita i reduir els temps d'espera per mitjà de la utilització de tècniques d'intel·ligència ambiental i Internet of things.

El projecte s'implementarà per mitjà del desenvolupament d'una aplicació mòbil Android i un mòdul complementari que permet integrar el programa desenvolupat amb les aplicacions clíniques existents. El sistema es podrà adaptar fàcilment a altres àmbits d'aplicació on el desplaçament a una cita siga un factor susceptible de millora: reunions de treball, servicis públics, tutories acadèmiques, etc.

Paraules clau: Internet of things, informàtica mèdica, gestió hospitalària, citació, Android.

Agradecimientos

Este trabajo ha sido posible gracias a la guía y los consejos de mis dos directores, Vicente Pelechano Ferragud y Andrés Boza García. Siguiendo sus indicaciones he sido capaz de plasmar en un proyecto consistente y orquestado la nube de ideas de las que partía en un principio cuando quería aprovechar los conocimientos recibidos en el Máster Universitario en Ingeniería Informática para darle una vuelta de tuerca a una parte de la asistencia sanitaria que creo que es posible mejorar.

Pero sobretodo, no hubiera sido capaz de llegar a finalizar esta pequeña locura que ha sido en mi vida estudiar este máster sin el apoyo y la paciencia de Ángela, la compañera de toda mi vida, y también de mi hija Julia, que se ha privado de muchos juegos y atenciones mientras solo veía una espalda delante de un ordenador.

A todos y cada uno de ellos les estoy inmensamente agradecido por haberme ayudado a lograr una de las metas más altas que me he planteado en la vida.

Tabla de contenidos

Índice de figuras

Índice de tablas

1. Introducción

1.1. Presentación

Me llevo dedicando de forma profesional a la informática médica desde el año 1997, y en este tiempo he llevado a cabo proyectos tanto para hospitales y clínicas privadas como para hospitales del sistema público de salud, entre los que se encuentran soluciones para numerosas especialidades médicas, pero sobretodo para ginecología y obstetricia, otorrinolaringología y pediatría. Esta actividad y el hecho de haber desempeñado siempre mis funciones en el interior de un hospital, en contacto con el personal sanitario y los pacientes, me ha llevado a conocer distintas estrategias y prácticas a la hora de gestionar un centro médico.

En unos años se ha pasado desde una situación en la que la tecnología se empezaba a introducir en el entorno sanitario hasta el momento actual, en el que los individuos están permanentemente conectados gracias a innumerables servicios de las tecnologías de la información y en el que los centros sanitarios basan su trabajo en sistemas de información integrales.

A pesar todo ello, hay un aspecto de la gestión de la asistencia sanitaria que sigue siendo esencialmente el mismo que en los albores de la medicina: la llegada y recepción del paciente a la consulta, es decir, todo lo que sucede desde que el paciente está cómodamente sentado en su casa hasta que está en manos del médico que va a tratar su dolencia.

Por todo ello, en el presente Trabajo de Final de Máster intento obtener una forma de mejorar ese aspecto de la asistencia sanitaria, que indudablemente es importante para el paciente ya que al malestar que pueda suponerle el motivo por el que acude hay que sumarle la tensión emocional que pudiera provocarle el desenvolverse en un medio potencialmente hostil y desconocido como puede serlo para él un centro sanitario.

1.2. Objetivos del proyecto

El objetivo de este proyecto es mejorar las acciones que se llevan a cabo desde que un paciente ha sido citado en una clínica hasta que es atendido por el médico, sirviéndose en lo posible de técnicas, disciplinas y tecnologías tratadas en el Máster Universitario en Ingeniería en Informática de la Universidad Politécnica de Valencia durante los cursos 2015-2016 y 2016-2017, y buscando una mayor calidad asistencial de forma que se mejore la satisfacción del paciente y la eficiencia del centro.

Este objetivo primero se concretará en el diseño e implementación de un sistema de información de naturaleza heterogénea que estará compuesto por los elementos software y hardware que se estimen oportunos (dispositivos móviles, servidores, estaciones de

trabajo, aplicaciones de escritorio, bases de datos, etc.) y que permitirán que se realicen de forma automática algunas de las acciones que actualmente se llevan a cabo de forma manual en el proceso que transcurre desde la cita de un paciente hasta su salida de la clínica después de haber sido atendido por el personal facultativo.

Esta automatización de acciones permitirá, bajo el punto de vista del paciente, tener a su disposición datos en tiempo real sobre aspectos tales como la ubicación de los aparcamientos cercanos y de la puntualidad del centro médico en su cita, para que le sea posible gestionar su hora de llegada, o invertir su tiempo en otra tarea en lugar de estar esperando.

Desde el punto de vista del centro médico, ofrecerá la posibilidad de tener acceso a más información sobre la ubicación de los pacientes y el momento de su llegada, y también reducir los costes derivados de no tener que dedicar su personal de administración o enfermería a recibir a los pacientes y darle las indicaciones oportunas, y también a los inherentes a la adquisición, instalación y mantenimiento de elementos tales como quioscos o pantallas para automatizar la recepción de los pacientes.

1.3. Estructura del proyecto

El presente proyecto se estructura en cinco bloques:

- En la primera parte, 1. Introducción, se presenta el proyecto y definen los objetivos que se persiguen.
- En la segunda parte, 2. Descripción del problema, se hace una primera aproximación al problema a resolver.
- En la tercera parte, 3. Contexto Tecnológico, se estudian y analizan las tecnologías que es necesario conocer y valorar para llevar a cabo este proyecto.
- En la cuarta parte, 4. Propuesta, se detalla paso a paso la solución encontrada al problema, desde el análisis hasta los detalles de la implementación.
- En la quinta parte, 5. Conclusiones. Líneas futuras, se hace un resumen de los resultados y sus aplicaciones, y se anticipan posibles ampliaciones.

2. Descripción del problema

Cuando una persona tiene que ir al médico tiene que llevar a cabo todo un ritual para que el facultativo trate su dolencia. Una vez que identifica la necesidad de concertar la cita, empieza un protocolo aceptado tácitamente tanto por el paciente como por el centro sanitario, y que tiene como objetivo dar al paciente la mejor asistencia posible y a la vez ofrecer sus servicios a todos los pacientes que lo soliciten de la forma más rápida y eficiente y sin dilatar en el tiempo las citas.

Hasta la fecha, los esfuerzos en mejorar todo el proceso siempre han ido encaminados a optimizar la planificación de la citación, aparte de algunos esfuerzos por mejorar la llamada al paciente que está en la sala de espera, pero se ha detectado que existen varias partes en el proceso que podría ser mejoradas mediante la utilización de técnicas que implican el uso de Tecnologías de la Información.

Estas partes susceptibles de mejora tienen además algo en común: si se pudieran mejorar añadirían valor a la experiencia del paciente en su visita al centro médico, cuya misión es buscar el bienestar del paciente y su pronta recuperación.

Para empezar a describir el problema es necesario primeramente aclarar los términos *hospital*, *centro médico*, *centro hospitalario*, etc. Para efectos del presente trabajo asumiremos que un *hospital* está compuesto por unidades funcionales independientes llamadas *clínicas*. Cada clínica tendrá su propia recepción y sala de espera, y sus médicos atenderán a los pacientes en salas llamadas *consultas*. Un hospital tendrá además una recepción principal pero esta no gestionará citas de pacientes, ya que toda cita será derivada a la recepción de una de las clínicas.

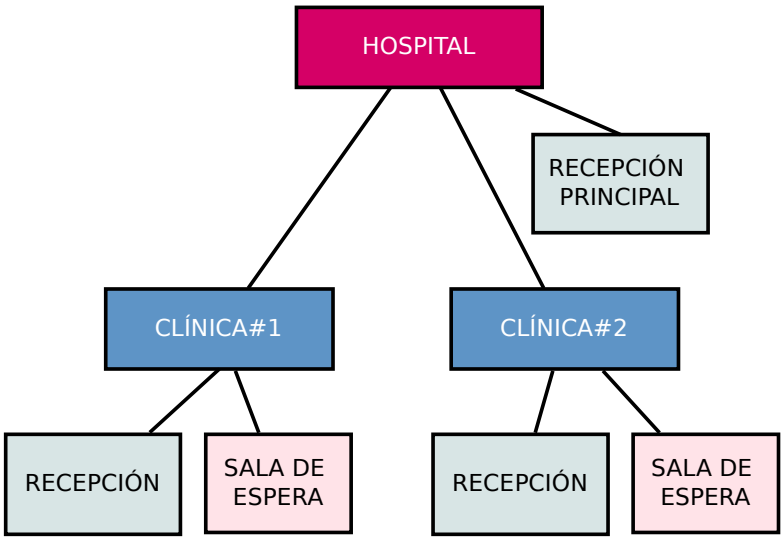


Fig. 1: Estructura organizativa de un hospital

El proceso para la petición de cita y la llegada de un paciente a una clínica consta normalmente de las siguientes 16 fases:

1. El paciente contacta con la clínica presencialmente, por medio de una llamada telefónica o por medio de un sistema de citación *online*.
2. El paciente negocia una cita con el personal de la clínica o con su sistema de citación automatizado.
3. La clínica proporciona al paciente una confirmación de la cita acordada, que puede ser de forma verbal, con una tarjeta donde aparece anotada la cita, o bien de forma electrónica con una pantalla de confirmación, un sms o un correo electrónico.
4. En algunos casos, con antelación a la cita la clínica envía al paciente un sms o un correo electrónico como recordatorio.
5. El día de la cita y a la hora prevista, el paciente acude a la clínica.
6. Dentro del centro sanitario, el paciente se dirige al hospital y deambula por su interior hasta llegar a la clínica.
7. El paciente anuncia su llegada, bien al personal de recepción o bien actuando sobre el sistema de información de la clínica por medio de un *quiosco* o un dispositivo similar.
8. En algunos casos, antes de ser atendido el paciente entrega a la clínica una contraprestación del servicio que va a recibir, bien por medio de la presentación de su tarjeta SIP (siglas de Sistema de Información Poblacional), de la tarjeta de su compañía aseguradora, o bien mediante el pago con tarjeta de crédito o en metálico.
9. El personal de recepción o el sistema de información de la clínica indica al paciente dónde debe esperar hasta ser atendido.
10. El paciente se dirige al lugar indicado y permanece allí hasta ser llamado a la consulta.
11. El paciente es llamado a la consulta, bien de forma verbal o por medio de algún sistema electrónico como pantallas o paneles de información.
12. El paciente pasa a la consulta y es atendido.
13. El paciente abandona la consulta.
14. En algunos casos, es en este punto donde se entrega a la clínica la contraprestación que se ha mencionado en la fase 8.
15. El paciente abandona la clínica.

Modelando este proceso según el estándar BPMN2 (Business Process Modeling Language 2) obtenemos el siguiente diagrama:

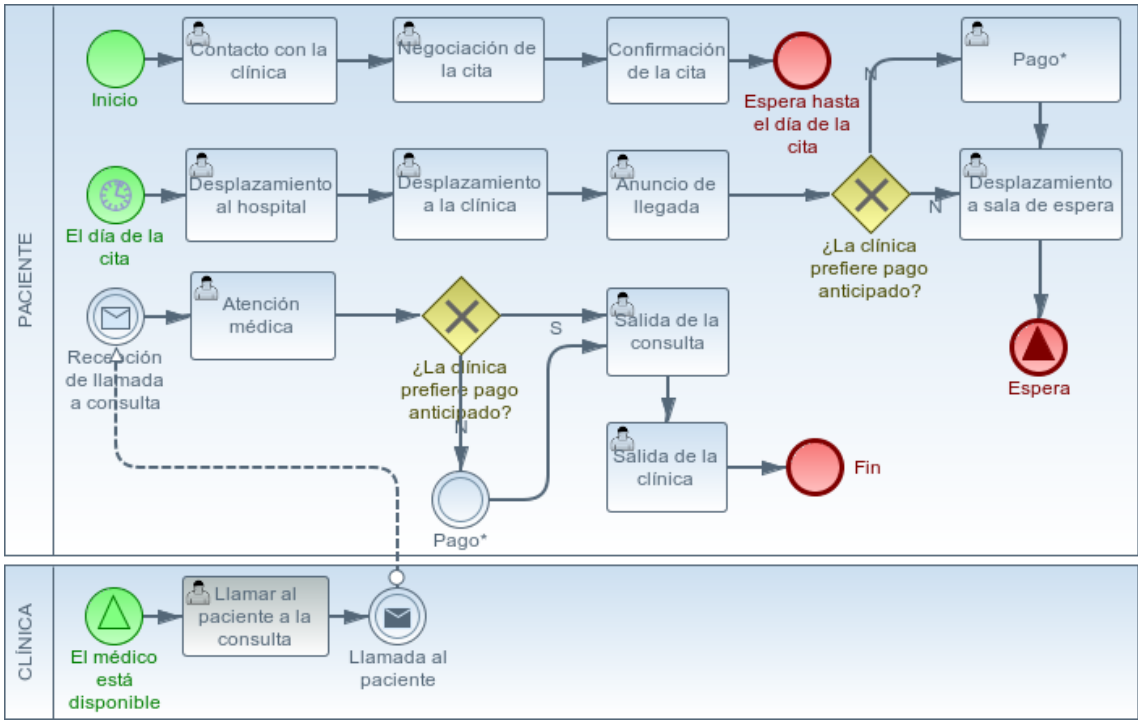


Fig. 2: Diagrama BPMN2 del proceso tradicional

* El término *Pago* se ha utilizado de forma genérica para referirse a la contraprestación que el paciente entrega a la clínica, y que se corresponde con la fase 8 descrita anteriormente.

Esta forma de proceder presenta los siguientes inconvenientes:

1. El paciente puede haber olvidado o extraviado la información de la cita: dónde y a qué hora debe presentarse.
2. La cita puede requerir la aportación por parte del paciente de documentación que puede haber olvidado.
3. El paciente se puede encontrar con diferentes circunstancias que pueden retrasarle: congestión de tráfico, búsqueda de aparcamiento, huelgas de transporte, etc.
4. El médico puede llevar retraso. Es habitual que en las clínicas se cite a los pacientes con una frecuencia algo inferior al necesario con el objetivo de maximizar la productividad, con lo que los retrasos se van acumulando de forma progresiva a lo largo de la jornada.
5. Hay médicos que también atienden urgencias o incluso partos mientras pasan consulta, con lo que se producen nuevos retrasos.

6. El paciente no es conocedor del retraso que lleva el médico hasta que no llega a la clínica, con lo que después del esfuerzo realizado por llegar a su hora se ve obligado a esperar.

7. No se suele facilitar al paciente información veraz y objetiva del tiempo que debe esperar a ser atendido.

8. Si en un momento dado un paciente no acude a su cita, la clínica no tiene ninguna información sobre la ubicación del paciente.

9. La clínica no tiene una forma eficaz de comprobar si el paciente sabe que tiene cita.

10. Cuando el paciente llega a una clínica, sobretodo por vez primera, no sabe a quién debe dirigirse para comunicar su llegada, con lo que se producirán nuevas esperas.

11. Un paciente no tiene una forma sencilla y eficaz de comunicar a la clínica la imposibilidad de acudir a una cita.

12. Cuando un paciente está en una sala de espera junto con otros pacientes, llega un momento en el que debe ser llamado a consulta, y para preservar la privacidad se debe llevar a cabo esta llamada de forma de forma discreta sin revelar ninguno de sus datos personales. Este aspecto es especialmente delicado en especialidades como cirugía plástica, urología, ginecología o psiquiatría.

13. El hecho de que pacientes con diversas patologías compartan una sala de espera durante un tiempo prolongado puede causarles problemas de estrés que se añaden a las dolencias propias de su problema médico.

14. Los retrasos producen en los pacientes insatisfacción y frustración, con lo que la sensación de calidad percibida es baja, independientemente de la efectividad del tratamiento médico.

15. El prolongar más de lo imprescindible la estancia de un paciente en un hospital aumenta de forma innecesaria el riesgo de que contraiga algún tipo de infección intrahospitalaria.

En los últimos años el procedimiento de citación se ha mejorado gracias a las tecnologías de la Información siguiendo varias estrategias, que se han venido aplicando bien de forma aislada o combinada:

1. Utilizar software de citación.

2. Enviar recordatorios de las citas mediante SMS.

3. Utilizar sistemas de citación online.

4. Instalar terminales para que los pacientes comuniquen su llegada.

5. Instalar pantallas para mostrar el paciente que debe entrar a consulta.

Todos estos esfuerzos van encaminados a mejorar algunas partes del proceso, concretamente:

- El software de citación se suele utilizar en las fases 2, 3, 8 y 14.
- El recordatorio automático se aplica en la fase 4.
- Los sistemas de citación online se aplican en la etapa 2 y 3.
- Los terminales para que el paciente comunique su llegada se aplican en la etapa 7.
- Las pantallas para indicar al paciente que pase a consulta se aplican en la etapa 11.

Con ello, hay etapas en las que apenas se suele intervenir desde el ámbito de las TI, que son las siguientes:

- Fase 5. Recorrido del paciente desde su lugar de origen al hospital
- Fase 6. Desplazamiento del paciente por las instalaciones del hospital
- Fase 7. Llegada del paciente a la clínica
- Fases 8 y 14. Monetización de la asistencia sanitaria
- Fase 9. Indicación al paciente de la ubicación de la sala de espera
- Fase 10. Deambulación del paciente desde la recepción a la sala de espera
- Fase 11. Llamada del paciente a consulta
- Fase 12. Entrada del paciente a la consulta y prestación de la atención médica
- Fase 13. Salida del paciente de la sala de consulta

Analizando estas últimas se puede apreciar que hay un cierto vacío en la asistencia al paciente desde el momento en que se dispone a dirigirse al hospital hasta su llegada a la clínica, dado que no se han encontrado soluciones destinadas a este fin.

En este proyecto se pretende desarrollar una aplicación que cubra precisamente esa necesidad y que además ayude a aumentar la eficacia de la clínica y a reducir el tiempo que permanece el paciente en el hospital, redundando todo ello en una mayor calidad de los servicios sanitarios que además será percibida por el paciente.

3. Contexto Tecnológico

Se pretende desarrollar un sistema de información que conecte la clínica con el paciente, teniendo en cuenta que la clínica tendrá su propio sistema de información y que se pretende proponer una solución con localización y con elementos propios de *Internet of Things*. Para ello será necesario utilizar las siguientes tecnologías:

1. Un entorno de programación compatible con el sistema de información de la clínica
2. Un entorno de programación para smartphones
3. Un sistema de paso de mensajes que haga de intermediario entre los dispositivos móviles y el sistema de información de la clínica, conectado a Internet y diseñado para funcionar con aplicaciones propias de *Internet of Things*
4. Un sistema de geolocalización
5. Un sistema de ubicación en interiores
6. Acceso a los datos disponibles en las plataformas Open Data de las *smart cities*

3.1. Entorno de programación para informática médica

En el entorno sanitario se utiliza una gran multitud de lenguajes de programación. Los requerimientos en este ámbito siempre priorizan la seguridad y la fiabilidad, así que normalmente se utilizan paradigmas de programación, lenguajes y sistemas de gestión de bases de datos que tienen un alto grado de madurez. Es posible encontrar productos desarrollados en Cobol, Java, Visual Basic, Delphi...

Para causar el menor impacto posible, se utilizará un sistema de paso de mensajes o de intercambio de datos que hará de intermediario entre el sistema de información de la clínica y el paciente.

Este intercambio de datos habitualmente se lleva a cabo de varias formas:

1. Accediendo directamente a la base de datos, normalmente a través de ODBC (Open Database Connectivity) o JDBC (Java Database Connectivity). Los sistemas de gestión de bases de datos más habituales son; MySQL, Oracle y Microsoft SQL Server
2. Mediante un servicio web o una API REST, utilizando el formato de texto plano, XML, JSON o YAML

3.1.1. MySQL

Según (DuBois, 2009), el origen de MySQL se remonta a 1979, a la herramienta de base de datos UNIREG, creada por Michael Widenius para la empresa sueca TcX. En 1993, TcX comenzó la búsqueda de un RDBMS (Relational Database Management System, sistema de gestión de bases de datos relacionales) con una interfaz SQL para desarrollar aplicaciones web. Probaron distintos servidores comerciales pero todos eran demasiado lentos par las tablas de gran tamaño de TcX. También se fijaron en mSQL pero carecía de determinadas funciones necesarias para TcX. Por ello Widenius comenzó a desarrollar un nuevo servidor. La interfaz de programación de desarrolló explícitamente como la utilizada por mSQL.

En 1995, TcX publicó MySQL en Internet. MySQL 3.11.1 se dio a conocer en 1996 como distribución binaria para Linux y Solaris. En la actualidad, MySQL funciona en muchas más plataformas y está disponible en formato binario y en código fuente.

La empresa MySQL AB se creó para ofrecer distribuciones de MySQL de código abierto y de licencia comercial, y para proporcionar asistencia técnica, servicios de monitorización y formación. En 2008, Sun Microsystems adquirió MySQL AB, siguiendo con el compromiso de código fuente; a su vez, Sun Microsystems fue comprada por Oracle Corporation en 2010. En ese momento Oracle ya poseía desde 2005 Innobase Oy, empresa finlandesa desarrolladora del motor InnoDB para MySQL.

Oracle provee el código fuente de MySQL Community Edition y versiones compiladas para diferentes sistemas operativos,24 aunque el rendimiento de MySQL se encuentra optimizado para sistemas GNU/Linux, con pequeñas diferencias de rendimiento entre las diferentes distribuciones.

Según (Wikipedia-MySQL, 2017), algunas de sus características son:

- MySQL es una base de datos muy rápida en la lectura cuando utiliza el motor no transaccional MyISAM, pero puede provocar problemas de integridad en entornos de alta concurrencia en la modificación.
- Funciona sobre múltiples plataformas, incluyendo: AIX, BSD, FreeBSD, HP-UX, Kurisu OS, GNU/Linux, Mac OS X, NetBSD, OpenBSD, OS/2 Warp, QNX, SGI IRIX, Solaris, SunOS, SCO OpenServer, SCO UnixWare, Tru64, eBD, Windows 95, Windows 98, Windows NT, Windows 2000, Windows XP, Windows Vista, Windows 7, Windows 8, Windows 10 y Windows Server (2000, 2003, 2008 y 2012) y OpenVMS.
- Usa GNU Automake, Autoconf, y Libtool para portabilidad.
- Usa multihilos mediante hilos del kernel.
- Usa tablas b-tree en disco.
- Utiliza tablas hash temporales en memoria.

- Ofrece un sistema de contraseñas y privilegios seguro mediante verificación basada en el host y el tráfico de contraseñas está cifrado al conectarse a un servidor.
- Soporta gran cantidad de datos (hasta 50 millones de registros).
- Se permiten hasta 64 índices por tabla (32 antes de MySQL 4.1.2). Cada índice puede consistir desde 1 hasta 16 columnas o partes de columnas. El máximo ancho de límite son 1000 bytes (500 antes de MySQL 4.1.2).
- Los clientes se conectan al servidor MySQL usando sockets TCP/IP en cualquier plataforma. En sistemas Windows se pueden conectar usando *named pipes* y en sistemas Unix usando ficheros socket Unix.
- En MySQL 5.0, los clientes y servidores Windows se pueden conectar usando memoria compartida.
- MySQL contiene su propio paquete de pruebas de rendimiento proporcionado con el código fuente.

3.1.2. Bases de datos Oracle

Según (Wikipedia-Oracle, 2017), Oracle Database es un sistema de gestión de base de datos de tipo ORDBMS (Object-Relational Data Base Management System, sistema de gestión de base de datos objeto-relacional), desarrollado por Oracle Corporation. habiendo sido certificadas las últimas versiones para poder trabajar bajo GNU/Linux.

La tecnología Oracle se encuentra prácticamente en todas las industrias alrededor del mundo y en las oficinas de 98 de las 100 empresas Fortune 100. Oracle es la primera compañía de software que desarrolla e implementa software para empresas cien por ciento activado por Internet a través de toda su línea de productos: base de datos, aplicaciones comerciales y herramientas de desarrollo de aplicaciones y soporte de decisiones. Oracle es el proveedor mundial líder de software para administración de información.

Según (Oracle, 2010), algunas de las características de Oracle (11g) son las siguientes:

- Recuperación ante fallos
- Reconstrucción de índices en línea
- Reorganización en línea de tablas organizadas por índice
- Cambios en línea del sistema - CPU, disco, memoria
- Recuperación de datos de medios en bloques
- Recuperación y backup en línea y paralelos
- Recuperación de espacios de tablas en un momento determinado
- Recuperación de prueba
- Administración automática de cargas de trabajo
- Auditoría detallada

- Compatibilidad con Java
- Servicios web de bases de datos
- Compatibilidad de XML en bases de datos
- Procedimientos y disparadores PL/SQL almacenados
- Compilación nativa de Java
- Compilación nativa de PL/SQL
- Integración con Active Directory
- Procedimientos .NET almacenados
- Administración automática de memoria
- Administración automática de almacenamiento
- Funciones analíticas SQL
- Replicación avanzada
- Transacciones/Consultas distribuidas

3.1.3. Microsoft SQL Server

Según (Wikipedia-SQLServer, 2017), Microsoft SQL Server es un sistema de manejo de bases de datos del modelo relacional, desarrollado por la empresa Microsoft Corporation.

El lenguaje de desarrollo utilizado (por línea de comandos o mediante la interfaz gráfica de Management Studio) es Transact-SQL (TSQL), una implementación del estándar ANSI del lenguaje SQL, utilizado para manipular y recuperar datos, crear tablas y definir relaciones entre ellas.

Dentro de los competidores más destacados de SQL Server están: Oracle, MariaDB, MySQL, PostgreSQL. SQL Server solo está disponible para sistemas operativos Windows de Microsoft.

Puede ser configurado para utilizar varias instancias en el mismo servidor físico, llevando generalmente la primera instalación el nombre del servidor, y las siguientes nombres específicos (con un guion invertido entre el nombre del servidor y el nombre de la instalación).

Sus principales características son:

- Soporta transacciones
- Soporta procedimientos almacenados
- Incluye un entorno gráfico de administración
- Permite trabajar en modo cliente-servidor, donde la información y datos se alojan en el servidor y los terminales o clientes de la red sólo acceden a la información
- Permite administrar información de otros servidores de datos

Este sistema incluye una versión reducida, llamada MSDE con el mismo motor de base de datos pero orientado a proyectos más pequeños, llamada SQL Express Edition, que se distribuye en forma gratuita.

3.2. Entornos de programación para *smartphones*

Hasta hace relativamente poco se disputaban el mercado de los sistemas operativos para dispositivos móviles las plataformas Android, iOS, Windows Phone y BlackBerry OS. Según datos de abril de 2017 de (Kantar, 2017), actualmente la cuota de mercado de Android en España es de un 92,0, por un 7,5% de iOS y con una presencia residual de Windows Phone y otros sistemas operativos.

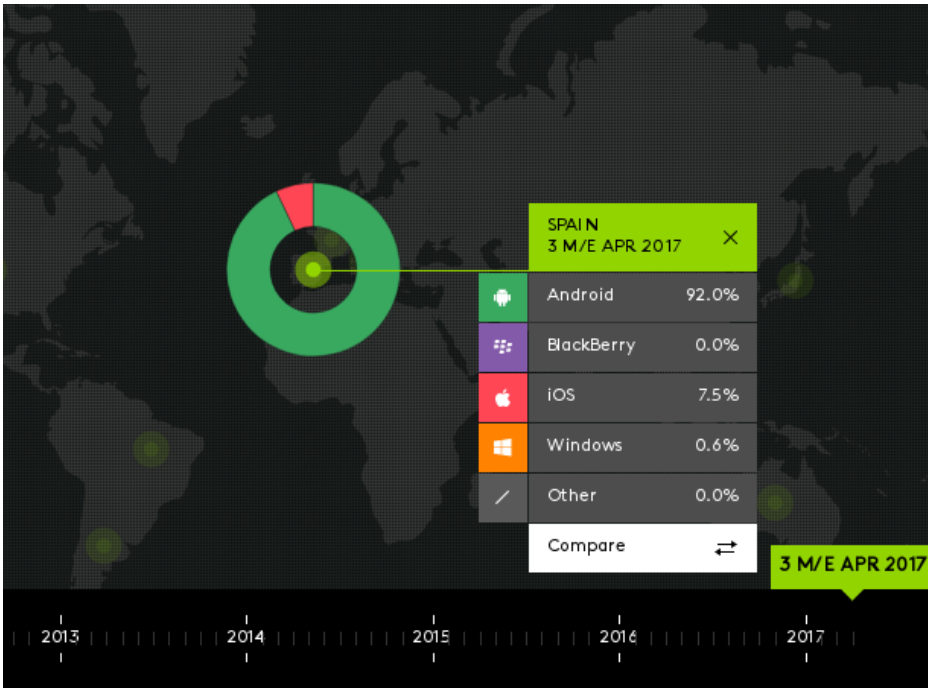


Fig. 3: Sistemas operativos móviles en España. Abril de 2017.
Imagen tomada de www.kantarworldpanel.com

3.2.1. Android. Android Studio

Android es un sistema operativo construido sobre Linux. Lo utilizan una enorme cantidad de dispositivos, desde smartphones y tabletas a consolas de videojuegos.

Sus características, según (Wikipedia-Android, 2017), son la siguientes:

- Es multitarea.
- Soporta SQLite, una base de datos ligera.
- Soporta las siguientes tecnologías de conectividad: GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, HSDPA, HSPA+, NFC y WI-MAX, GPRS, UMTS y HSDPA+

- El navegador web incluido está basado en el motor de renderizado de código abierto WebKit, emparejado con el motor JavaScript V8 de Google Chrome.
- Aunque la mayoría de las aplicaciones están escritas en Java, no hay una máquina virtual Java en la plataforma. El bytecode Java no es ejecutado, sino que primero se compila en un ejecutable Dalvik y se ejecuta en la máquina virtual Dalvik. Dalvik es una máquina virtual especializada que está diseñada específicamente para Android y optimizada para dispositivos móviles que funcionan con batería y que tienen memoria y procesador limitados.
- Soporta los siguientes formatos multimedia: WebM, H.263, H.264, MPEG-4 SP, AMR, AMR-WB , AAC, HE-AAC , MP3, MIDI,Ogg Vorbis, WAV, JPEG, PNG, GIF y BMP.
- Soporta cámaras de fotos, de vídeo, pantallas táctiles, GPS, acelerómetros, giroscopios, magnetómetros, sensores de proximidad y de presión, sensores de luz, gamepad, termómetro, aceleración por GPU 2D y 3D.
- Utiliza Google Play, un catálogo de aplicaciones gratuitas o de pago en el que pueden ser descargadas e instaladas en dispositivos Android sin la necesidad de un ordenador convencional.
- Tiene soporte nativo para pantallas capacitivas con soporte multi-táctil.
- Ofrece soporte completo para Bluetooth.
- Soporta videollamada a través de Google Hangouts.

El entorno de desarrollo oficial y recomendado para Android es Android Studio, aunque también se utilizar Eclipse con los plugins adecuados.

Sus características son la siguientes, según (Android.com, 2017):

- Incorpora un editor de código inteligente.
- Su sistema de construcción es robusto y flexible.
- Está optimizado para todos los dispositivos Android.
- Tiene la característica *Instant Run*, que actualiza el código y los cambios en los recursos en la aplicación mientras se está ejecutando.
- Tiene un emulador rápido y completo en funcionalidades.
- Está diseñado para equipos de trabajo. Integración con Git, Subversion, etc.

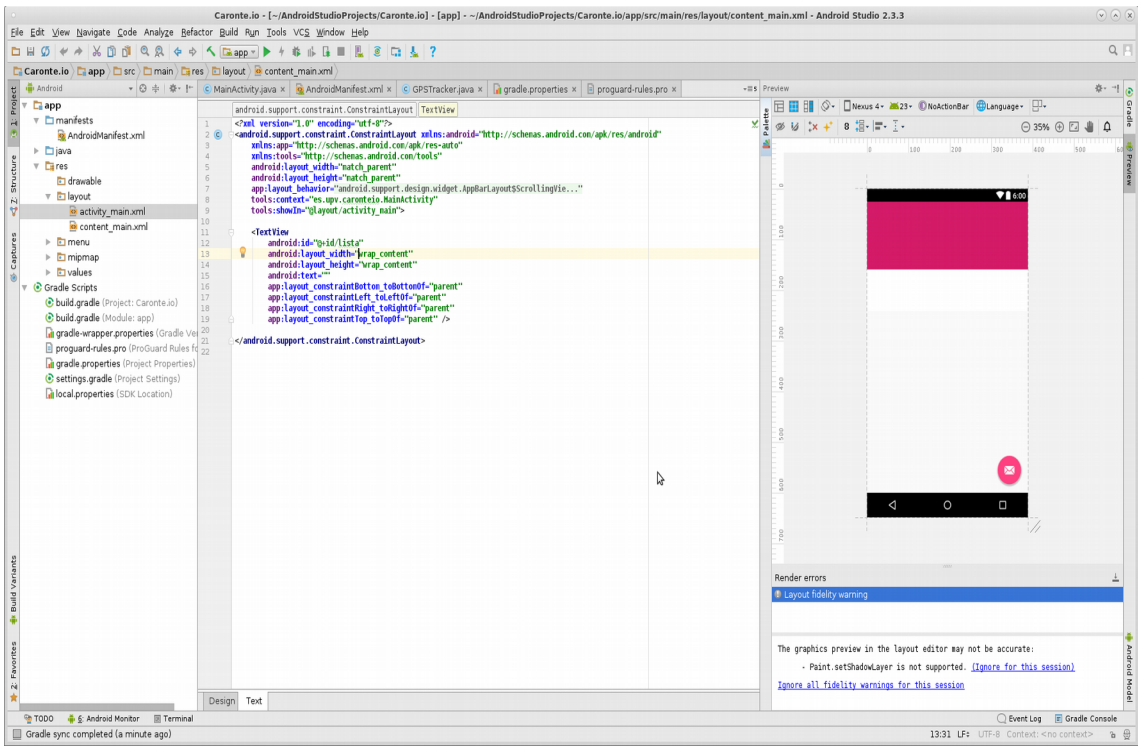


Fig. 4: Android Studio

3.2.2. iOS. Xcode IDE

Según (Wikipedia-iOS, 2017), iOS es un sistema operativo móvil de la multinacional Apple Inc. Originalmente desarrollado para el iPhone (iPhone OS), después se ha usado en dispositivos como el iPod Touch y el iPad, y no permite su instalación en hardware de terceros.

iOS se deriva de OSX, que a su vez está basado en Darwin BSD, y por lo tanto es un sistema operativo tipo Unix. Cuenta con cuatro capas de abstracción: la capa del núcleo del sistema operativo, la de servicios principales, la de medios y la de *Cocoa Touch*.

Algunas de sus características son las siguientes:

- Soporta multitarea de forma opcional: Antes de iOS 4, la multitarea estaba reservada para aplicaciones por defecto del sistema. La multitarea sólo es compatible desde el iPhone 3GS, iPad 1, iPod Touch (3ª generación). A partir de iOS 4, dispositivos de tercera generación y posteriores permiten el uso de 7 APIs para multitarea, específicamente: audio en segundo plano, voz IP, localización en segundo plano, notificaciones locales y completado de tareas.
- Las versiones anteriores a iOS 8 no permiten el uso de la Plataforma Java y Adobe Flash. IOS usa HTML5 como una alternativa a Flash.
- Su interfaz gráfica tiene capacidad para gestos multitáctiles.
- Soporta acelerómetros internos.

La plataforma oficial para crear aplicaciones para iOS es Xcode IDE.

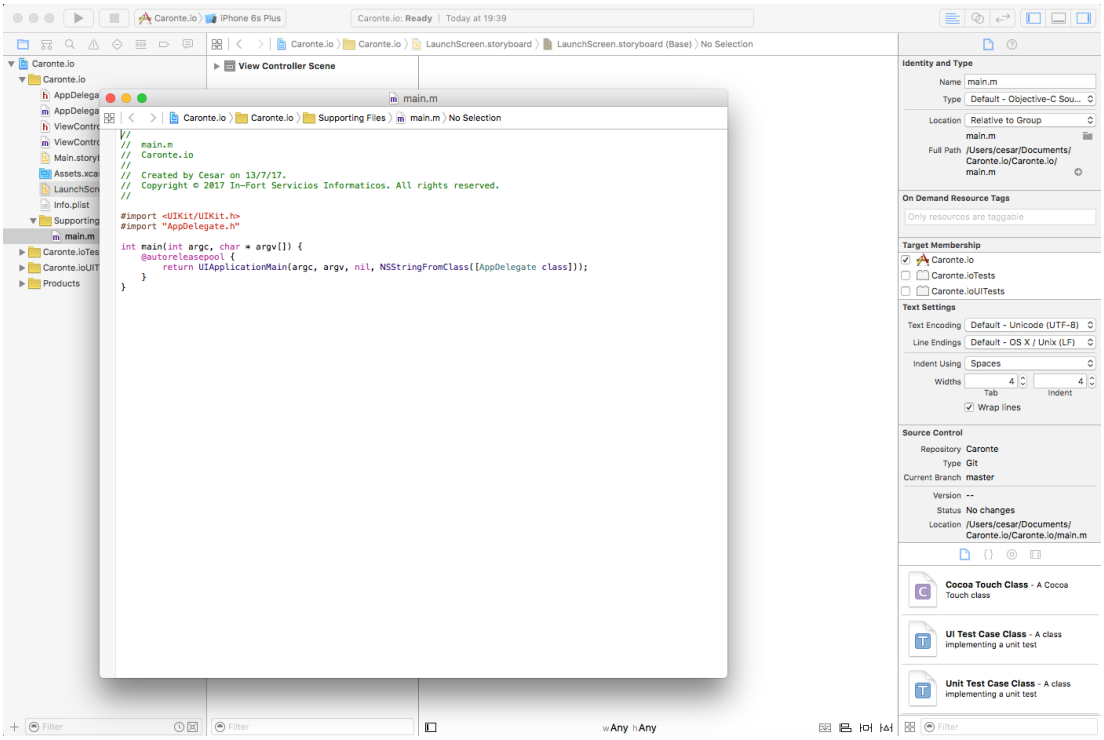


Fig. 5: Xcode IDE

Según (Apple.com, 2017), las caracteres de Xcode IDE son las siguientes:

- Editor de código profesional, con *code completion* avanzada
- Editor asistente
- Editor de versiones
- Catálogo de *assets*
- Constructor de interfaces integrados
- Simulador
- Sistema de construcción integrado
- Compiladores para C, C++ y Objective-C
- Depurador gráfico
- Integración continua
- Sugerencias de código
- Análisis estático

3.3. Sistemas intermedios para IoT

En el contexto de Internet of Things, para compartir información entre sistemas heterogéneos, que es el caso más habitual, se utilizan protocolos de mensajería. Los más utilizados son los servicios web basados en APIs REST y el protocolo MQTT, y los formatos de fichero más utilizados son el texto plano, XML, JSON y YAML.

3.3.1. Servicios web

Según (Wikipedia-Servicios Web, 2017), un servicio web (en inglés, web service o web services) es una tecnología que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de ordenadores como Internet. La interoperabilidad se consigue mediante la adopción de estándares abiertos. Las organizaciones OASIS y W3C son los comités responsables de la arquitectura y reglamentación de los servicios Web.

Para mejorar la interoperabilidad entre distintas implementaciones de servicios Web se ha creado el organismo WS-I, encargado de desarrollar diversos perfiles para definir de manera más exhaustiva estos estándares.

El concepto básico es que un servicio web en una máquina atiende las peticiones de los clientes web y les envía los recursos solicitados.

Entre sus ventajas se encuentran:

- Aportan interoperabilidad.
- Fomentan estándares y formatos basados en texto, que son más sencillos de entender e implementar.
- Permiten la combinación de servicios y software en ubicados en distintos lugares geográficos de forma que se puedan producir nuevos servicios integrados.

Entre sus desventajas se encuentran:

- Su grado de desarrollo para llevar a cabo transacciones es inferior al de los estándares abiertos de computación distribuida como CORBA (Common Object Request Broker Architecture, DCOM (Distributed Component Object Model) o RMI (Remote Method Invocation).
- Ofrecen un bajo rendimiento comparado con los estándares anteriores
- Presentan problemas de seguridad al utilizar HTTP, servicio que está normalmente abierto en los cortafuegos.

3.3.2. REST

Según (Wikipedia-REST, 2017), la Transferencia de Estado Figurativo (en inglés Representational State Transfer o REST) es un estilo de arquitectura software para sistemas hipermedia distribuidos como la World Wide Web. El término se originó en el año 2000, en una tesis doctoral sobre la web escrita por Roy Fielding, uno de los principales

autores de la especificación del protocolo HTTP y ha pasado a ser ampliamente utilizado por la comunidad de desarrollo.

En (Marques, 2013) se afirma que se trata de un tipo de arquitectura de desarrollo web que se apoya totalmente en HTTP. Nos permite crear servicios y aplicaciones que pueden ser usadas por cualquier dispositivo que soporte el protocolo HTTP.

En la arquitectura REST, los URL (Universal Resource Locator) se estructuran de la forma:

```
{protocolo}://{dominio o hostname}[:puerto (opcional)]/{ruta del recurso}?
{consulta de filtrado}
```

Así pues, `http://www.cesarfort.com/tfm/` sería por ejemplo el URL para visualizar el trabajo de final de máster de la página personal de un alumno llamado Cesar Fort.

Para manipular los recursos, REST consta de los siguientes métodos:

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

3.3.3. MQTT. MQTT sobre WebSockets

Según (IBM, 2010), MQTT (Message Queue Telemetry Transport, transporte de colas de mensajes de telemetría) es un protocolo del estilo publicar/subscribir basado en brokers y ligero, diseñado para ser abierto, simple, ligero y fácil de implementar. Estas características lo hacen ideal para el uso en entornos restringidos, por ejemplo, pero no limitado a cuando la red es cara tiene un bajo ancho de banda o no es fiable, o a funcionar en un dispositivo empotrado con recursos limitados de procesador o memoria.

Las características del protocolo incluyen:

- El patrón de mensajes publicar/suscribir para proporcionar la distribución de mensajes uno a muchos y el desacoplamiento de aplicaciones.
- Un transporte de mensajes que es agnóstico al contenido del *payload* del mensaje.
- El uso de TCP/IP para proporcionar la conectividad de red básica.
- Tres niveles de calidad de servicio para entrega de mensajes:
    - **Como mucho una vez**, en la cual los mensajes se reparten según a los mejores esfuerzos de la red TCP/IP subyacente. Puede ocurrir pérdida o duplicación de mensajes. Este nivel se podría usar, por ejemplo, con datos de sensores ambientales donde no importa si una lectura individual se pierde dado que la siguiente se publicará a continuación.

  - **Por lo menos una vez**, donde se asegura la llegada de los mensajes, pero pueden haber duplicados.
  - **Exactamente una vez**, donde se asegura que los mensajes llegarán exactamente una vez. Este nivel se podría usar, por ejemplo, con sistemas de facturación en los cuales los mensajes perdidos o duplicados podrían llevar a aplicar cargos incorrectos.

- Una pequeña sobrecarga de transporte (la cabecera de longitud fija es de solo 2 bytes), a intercambios minimizados en el protocolo para reducir el tráfico de red.
- Un mecanismo para notificar de una desconexión anormal a las partes interesadas a una desconexión anormal utilizando las características de Últimas Voluntades y Testamento. El mecanismo de Últimas Voluntades y Testamento, según (HiveMQ, 2017), consiste en que cada cliente puede especificar su mensaje de últimas voluntades (un mensaje MQTT normal) cuando se conecta a un broker. El broker almacenará el mensaje hasta que detecte que el cliente se ha desconectado de forma abrupta. Si es así, el broker envía el mensaje a todos los clientes suscritos al tema, que fue previamente especificado en el mensaje de últimas voluntades. El mensaje de últimas voluntades y testamento se descarta si el cliente se desconecta de la forma normal enviado un mensaje DISCONNECT.

Los mensajes MQTT se organizan en *topics*(asuntos), a los que también se les suele llamar *temas* o *colas*, que será el término que más se utilice en el presente documento.

A continuación veremos el protocolo MQTT sobre WebSockets y tres implementaciones de MQTT: HiveMQ, Mosquitto y AWS IoT.

3.3.3.1. MQTT sobre WebSockets

Según se recoge en (HiveMQ, 2017-2), MQTT sobre WebSockets permite a un navegador hacer uso de todas las funcionalidades de MQTT y ello puede ser utilizado para, por ejemplo:

- Mostrar información de un dispositivo o sensor en tiempo real
- Recibir notificaciones, por ejemplo si hay una alerta o una condición crítica
- Consultar el estado de dispositivos con últimas voluntades y testamento (LWT) y mensajes retenidos
- Comunicarse eficientemente con una aplicación web móvil

WebSockets es un protocolo de red que proporciona comunicación bidireccional entre un navegador y un servidor web. Se estandarizó en 2011 y todos los navegadores modernos lo soportan de forma nativa. Del mismo modo de MQTT, el protocolo WebSockets está basado en TCP.

Cuando se utiliza el término MQTT sobre WebSockets, un mensaje, por ejemplo CONECTAR o PUBLICAR, se encapsula en uno o varios frames de WebSockets al transferirlo sobre la red. El protocolo WebSockets es adecuado como protocolo de transporte para MQTT porque la comunicación es bidireccional, ordenada y sin pérdida. Para comunicarse con un broker MQTT sobre WebSockets, el broker debe soportar WebSockets de forma nativa.

Existen numerosas plataformas software que implementan servidores MQTT. De entre ellas haremos un breve repaso a tres: HiveMQ, Mosquitto y AWS IoT.

3.3.3.2. HiveMQ

HiveMQ es, según (HiveMQ, 2017-3), un broker MQTT desarrollado por dc-Square GmbH, una empresa fundada en 2012 y con base en el sur de Alemania.

Fig. 6: HiveMQ
Imagen tomada de http://www.hivemq.com/blog/mqtt/

HiveMQ dispone de las siguientes características:

- Escalable. Ha sido construido para soportar cuentos de miles de dispositivos con un rendimiento extremo en la transferencia de mensajes.
- Seguro. HiveMQ ha sido construido para desarrollos críticos con los más altos requisitos en cuento a seguridad. La autentificación y la autorización son elementos clave en todo desarrollo MQTT seguro, y el sistema soporta incluso los más avanzados mecanismos de seguridad para proporcionar protección frente a clientes MQTT maliciosos. Se ofrece soporte para estándares como SSL/TLS para hacer que los datos permanezcan seguros mientras son transferidos desde y hacia HiveMQ.
- Simple. Es simple de usar, monitorizar y mantener. HiveMQ se puede descargar y arrancar en minutos para su evaluación. Para entornos de producción, HiveMQ proporciona scripts de arranque para los sistemas operativos más comunes.
- HiveMQ se puede descargar sin coste para testearlo y evaluarlo con un máximo de 25 conexiones, pero para entornos de producción es necesario adquirir una licencia comercial.

Es posible instalar HiveMQ en sistemas operativos Windows y también en sistemas basados en Unix: Linux,BSD, MacOS X y el propio Unix.

3.3.3.3. Mosquitto

Mosquitto es, según (Mosquitto, 2017), un broker de mensajes que implementa las versiones 3.1 y 3.1.1 del protocolo MQTT. Proporciona un método ligero de gestionar mensajes utilizando un modelo del tipo publicar/suscribir. Esto lo hace adecuado para la mensajería de "Internet of Things", como sensores de baja potencia o dispositivos móviles como smartphones, ordenadores embebidos o microcontroladores como Arduino. Se trata de un proyecto perteneciente a iot.eclipse.org.

Fig. 7: Mosquitto
Imagen tomada de https://projects.eclipse.org/free-tags/mqtt

El propósito de Mosquitto, como se afirma en (Eclipse, 2017), es proporcionar una pequeña implementación de un servidor de los protocolos MQTT y MQTT-SN (MQTT for Sensor Networks). El significado que se le pretende dar al término *pequeña es*:

1. Que solo se incluye la funcionalidad necesaria. Se puede utilizar compilación condicional para permitir omitir funcionalidades para una aplicación particular.
2. Que la funcionalidad está codificada de la forma más eficiente posible.
3. Que la parte exterior al sistema es lo más simple posible para la funcionalidad proporcionada.

El servidor tiene las siguientes características que no están descritas en la especificación de MQTT:

1. Un *bridge* MQTT, para permitir a Mosquitto conectarlo a otros servidores MQTT
2. Asegurar la comunicaciones mediante SSL/TLS
3. Autorización de usuarios: La posibilidad de restringir a determinados usuarios el acceso a ciertas colas de mensajes

Mosquitto está disponible para las siguientes plataformas: Microsoft Windows (a partir de Windows XP), Arch Linux, CentOS, Debian, Fedora, FreeBSD, Gentoo, openSUSE, OpenWrt, Raspberry Pi, Redhat Enterprise Linux, Slackware, Ubuntu, QNX e iOS.

3.3.3.4. AWS IoT

Según (Amazon, 2017), AWS IoT es una plataforma de nube administrada que permite a los dispositivos conectados interactuar con facilidad y seguridad con las aplicaciones en la nube y otros dispositivos. AWS IoT admite miles de millones de dispositivos y billones de mensajes, y es capaz de procesar y enrutar dichos mensajes a puntos de enlace de AWS y a otros dispositivos de manera fiable y segura. Con AWS IoT, las aplicaciones pueden realizar un seguimiento de todos los dispositivos y comunicarse con ellos en todo momento, incluso cuando no están conectados.

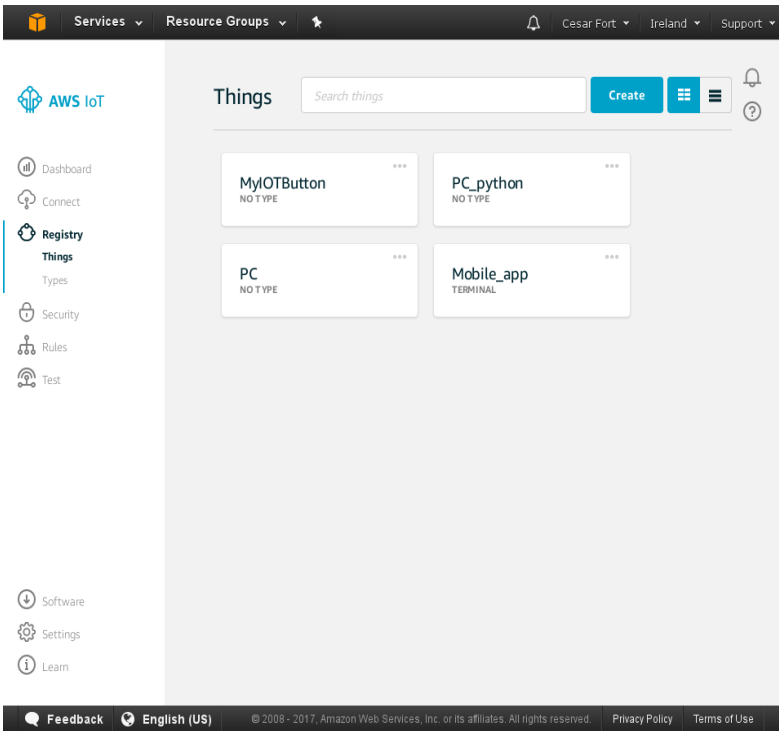


Fig. 8: AWS IoT. Inventario de objetos

AWS IoT facilita la utilización de servicios de AWS como AWS Lambda, Amazon Kinesis, Amazon S3, Amazon Machine Learning y Amazon DynamoDB, Amazon CloudWatch, AWS CloudTrail y Amazon Elasticsearch Service con la integración incorporada de Kibana para crear aplicaciones de IoT que recopilen, procesen, analicen y utilicen datos generados por dispositivos conectados sin necesidad de tener que administrar ninguna infraestructura.

3.3.4. XML

SGML (Standard Generalized Markup Languaje) fue uno de los primeros intentos en el que se pretendía combinar un formato de datos universalmente intercambiables con la posibilidad de almacenar los datos con información acerca de su presentación y formato. Este es un lenguaje basado en texto que utiliza datos de marcas, es decir, metadatos añadidos, como modo de autodescripción, y fue diseñado con la idea de que se transformara en un estándar para el marcaje de datos.

La aplicación más conocida de SGML es HTML (Hypertext Markup Languaje). Como las reglas para la creación de documentos SGML están bien establecidas, y como SGML se ha estado aplicando de manera intensiva en los sistemas de gestión de documentos, resultó simple crear un vocabulario específico HTML para que se convirtiera en un lenguaje de marcas universal para la visualización de información y para el enlace de las diferentes partes de la información.

Sin embargo, SGML es un lenguaje tan complicado que no resulta adecuado para el intercambio de datos en la web, y aunque HTML ha tenido un éxito enorme, tiene un ámbito muy limitado: su objetivo es tan solo lograr que se visualicen los documentos en un navegador.

XML es un subconjunto de SGML, con los mismo objetivos (marcas para describir los tipos de datos), pero sin el grado de complejidad del lenguaje original. XML fue diseñado para que sea fácilmente compatible con SGML, lo que implica que cualquier documento que sigue las reglas de sintaxis de XML es también, por definición, un documento SGML. XML describe una sintaxis que se utiliza para crear nuestros propios lenguajes.

Por ejemplo, supongamos que tenemos datos acerca del nombre de una persona y que queremos compartir esa información con otros. En lugar de crear un archivo de textos con este contenido:

```
Julia Fort
```

... o un archivo HTML como este:

```
<!DOCTYPE html>
<html lang="es">
    <head>
          <meta charset="utf-8" />
          <title>Nombre</title>
    </head>
    <body>
          <p>Julia Fort</p>
    </body>
</html>
```

... podríamos tener un archivo XML como este:

```
<nombreyapellidos>
    <nombre>Julia</nombre>
    <apellidos>Fort</apellidos>
</nombreyapellidos>
```

Con este ejemplo se muestra por qué los lenguajes de marcas se consideran autodescriptivos.

3.3.5. JSON

Según (ECMA, 2013), JSON (JavaScript Object Notation) es un formato de texto que facilita el intercambio de datos estructurados entre todos los lenguajes de programación. JSON posee una sintaxis de llaves, corchetes, dos puntos y comas y comas simples que son útiles en muchos contextos.

JSON fue inspirado por los literales de objeto de JavaScript, también conocidos como ECMAScript.

JSON es agnóstico en cuestión de números. En cualquier lenguaje de programación puede haber una gran variedad de tipos numéricos de varias capacidades y complementos, fijos o de coma flotante, binarios o decimales. Eso puede hacer difícil el intercambio entre diferentes lenguajes de programación; JSON, en lugar de ello, ofrece solo la representación de números que usan los humanos: una secuencia de dígitos, bajo la idea de que en todos los lenguajes de programación tienen sentido las secuencias de dígitos.

El texto en JSON es una secuencia de caracteres Unicode. En cuanto a los modelos de objeto soportados por muchos lenguajes de programación, JSON en su lugar ofrece una notación simple para expresar colecciones de pares nombre/valor. Esto puede ser interpretado por muchos lenguajes de programación y pueden convertirlos en estructuras de datos más complejas como registros, funciones, etc.

JSON también ofrece soporte para listas ordenadas de valores. Todos los lenguajes de programación tienen algún elemento para representar esas listas, como arrays, vectores o listas. Gracias a que estas estructuras se pueden anidar, JSON permite intercambiar complejas estructuras de datos entre lenguajes de programación *a priori* incompatibles.

3.3.6. YAML

Según (Ben-Kiki, 2009), YAML (YAML Ain't Markup Language, YAML no es un lenguaje de marcado), es un lenguaje de serialización de datos diseñado para ser amigable a los seres humanos que funciona bien con los lenguajes de programación para tareas comunes.

Las herramientas abiertas, interoperables y fáciles de entender han hecho avanzar a la informática inmensamente. Así, YAML fue diseñado desde el principio para ser útil y amigable a la gente que trabaja con datos.

YAML utiliza caracteres Unicode, algunos de los cuales proporcionan información estructural, y el resto contienen los datos.

Además, YAML consigue una limpieza única a través de la minimización de la cantidad de caracteres estructurales y permitiendo a los datos mostrarse a sí mismos de una forma natural y llena de significado. Por ejemplo, puede usarse indentación para crear pares estructurales `clave:valor` y guiones para crear viñetas.

Hay muchísimos tipos de estructuras de datos, pero todos se pueden representar adecuadamente con tres primitivas básicas:

- Funciones: hashes o diccionarios
- Secuencias: matrices o listas
- Escalares: cadenas o números

YAML hace uso de estas primitivas, y añade un sistema simple de tipado y de renombrado para formar un lenguaje completo para serializar cualquier estructura de datos nativa.

Mientras la mayoría de lenguajes de programación puede usar YAML para serialización de datos, YAML destaca trabajando los lenguajes que están construidos sobre los tres principios básicos. Esta categoría incluye lenguajes como Perl, Python. PHP, Ruby y Javascript.

Hay miles de lenguajes de programación diferentes, pero solo existen unos pocos para almacenar y transferir datos. Incluso aunque su potencial es potencialmente ilimitado, YAML fue creado específicamente para trabajar bien con casos de uso habituales como archivos de configuración, logs, mensajes entre procesos, compartición de datos entre lenguajes, persistencia de objetos y depuración de estructuras de datos complejas.

La idea que subyacente es que cuando es fácil ver y entender los datos, la programación se convierte en una tarea simple.

Los objetivos de diseño de YAML son:

1. Ser fácilmente legible por los humanos
2. Ser portable entre lenguajes de programación
3. Combinarse con las estructuras de datos nativas de los lenguajes ágiles
4. Tener un modelo consistente para apoyar herramientas genéricas
5. Permitir un procesamiento de un paso
6. Ser expresivo y extensible
7. Ser fácil de implementar y usar

Por ejemplo, esta sería una estructura de datos válida en YAML

```
Julia Fort: {
    Nombre: Julia,
    Apellido: Fort
    }
```

3.4. Geolocalización.

En este proyecto será necesario contar con un sistema de ubicación y localización geográfica para situar los dispositivos móviles y ubicarlos en la superficie de La Tierra. De entre estos sistemas el más conocido es el Sistema de Posicionamiento Global (o GPS, por sus siglas en inglés, Global Positioning System), pero no es el único. Existen otros, como GLONASS, BEIDOU y GALILEO.

3.4.1. GPS

Según (Gps, 2017), el Sistema de Posicionamiento Global (GPS) es un sistema de radionavegación de los Estados Unidos de América, basado en el espacio, que proporciona servicios fiables de posicionamiento, navegación, y cronometría gratuita e ininterrumpidamente a usuarios civiles en todo el mundo. A todo el que cuente con un receptor del GPS, el sistema le proporciona su localización y la hora exacta en cualesquier condición atmosférica, de día o de noche, en cualquier lugar del mundo y sin límite al número de usuarios simultáneos.

El GPS se compone de tres elementos: los satélites en órbita alrededor de la Tierra, las estaciones terrestres de seguimiento y control, y los receptores del GPS propiedad de los usuarios.

Desde el espacio, los satélites del GPS transmiten señales que reciben e identifican los receptores del GPS; ellos, a su vez, proporcionan por separado sus coordenadas tridimensionales de latitud, longitud y altitud, así como la hora local precisa.

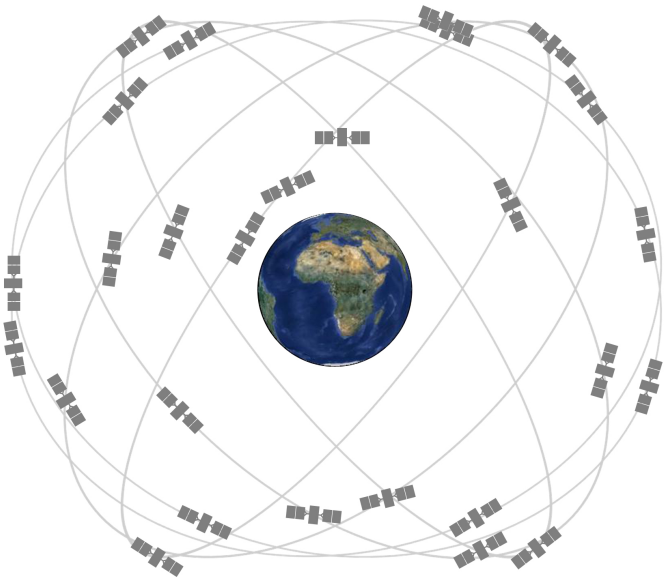


Fig. 9: GPS.
Imagen tomada de www.gps.gov
(http://www.gps.gov/multimedia/images/constellation.jpg)

Según (Inegi, 2017), el Sistema de Posicionamiento Global (GPS) es una constelación de satélites de navegación que orbitan alrededor de la Tierra a una altitud de cerca de

20.200 Km. A esta altitud, los satélites completan dos órbitas en un poco menos de un día.

Aunque originalmente diseñado por el Departamento de Defensa de EE.UU. para aplicaciones militares, su gobierno federal hizo el sistema disponible para usos civiles y levantó las medidas de seguridad diseñadas para restringir la precisión hasta 10m.

La constelación óptima consiste en 21 satélites operativos con 3 de repuesto. A partir de julio de 2006, había 29 satélites operacionales de la constelación.

Los satélites del GPS transmiten dos señales de radio de baja potencia, llamadas "L1" y "L2". Cada señal GPS contiene tres componentes de información: un código pseudoaleatorio, los datos de efemérides y datos de fecha y hora. El código pseudoaleatorio identifica al satélite que transmite su señal. Los datos de fecha y hora proporcionan información sobre la ubicación del satélite en cualquier momento, así como información sobre el estado del satélite y la fecha y hora actuales.

Para cada satélite, el tiempo es controlado por los relojes atómicos que lleva a bordo y que son cruciales para conocer su posición exacta.

Aunque el GPS puede dar posiciones muy precisas, aún hay fuentes de error. Estos incluyen los errores del reloj, los retrasos atmosféricos, sin saber exactamente dónde están los satélites en sus órbitas, las señales que se refleja de los objetos en la superficie de la Tierra, e incluso la degradación de la señal del satélite.

3.4.2. Otros sistemas de geolocalización

Según (Wikipedia-GLONASS, 2017), GLONASS (ГЛОНАСС, ГЛОбальная НАвигационная Спутниковая Система, sistema de navegación global por satélite) es el sistema de navegación desarrollado por la Unión Soviética, siendo hoy administrado por la Federación Rusa y que constituye el homólogo del GPS estadounidense y del Galileo europeo. El sistema está a cargo del Ministerio de Defensa de la Federación Rusa y los satélites se han lanzado desde Baikonur, en Kazajistán.

La constelación espacial del sistema de navegación por satélite BeiDou, abreviado BDS , según (Beidou, 2016), es el sistema de navegación desarrollado por la República Popular China, de funcionalidades análogas a los otros sistemas de navegación y con su propia constelación de satélites.

Según (Gsa, 2017), GNSS (Global Satellite Navigation System, sistema global de navegación por satélite), es el sistema de navegación europeo que proporciona información mejorada de posicionamiento y temporización con significantes implicaciones positivas para muchos servicios y usuarios europeos.

La independencia europea es uno de los objetivos principales del programa, pero Galileo también proporciona a Europa un asiento en la navegación global por satélite. El programa se diseño para ser compatible con todos los GNSSs actuales y futuros, e interoperable con GPS y GLONASS.

3.5. Localización de corto alcance

Los dos mayores inconvenientes que tienen los sistemas de localización satelitales son su precisión y el que no funcionan bien en interiores. Es por esto que para este cometido se suelen utilizar otras tecnologías. Las dos más utilizadas en la actualidad son NFC (Near Field Communication) y los beacons.

3.5.1. NFC

Según (Wikipedia-NFC, 2017), Near Field Communication (NFC, comunicación de campo cercano en español) es una tecnología de comunicación inalámbrica, de corto alcance y alta frecuencia que permite el intercambio de datos entre dispositivos.

Los estándares de NFC cubren protocolos de comunicación y formatos de intercambio de datos, y están basados en ISO 14443 (RFID, Radio-Frequency Identification) y FeliCa, un sistema de comunicación por radiofrecuencia para tarjetas inteligentes desarrollado por Sony Corporation.

Esta tecnología está siendo ampliamente utilizada por empresas para hacer compras de bienes y servicios, como por ejemplo, el servicio Apple Pay de Apple. Estas incluyen soporte para servicios de medios de pago, programas de fidelización a clientes, abonos de transporte público y demás. Estas apps a su vez, interactúan con las aplicaciones NFC del elemento seguro (tarjeta SIM) dando como resultado los servicios NFC.

Fig. 10: Logotipo de NFC
Imagen tomada de nfc.today
(http://nfc.today/images/NFC-Forum-NFCW.png)

También se puede usar, al igual que el Bluetooth como protocolo para transferir archivos de un teléfono a otro, siendo incluso más veloz que este sistema.

Soporta dos modos de funcionamiento, debiendo soportar ambos todos los dispositivos del estándar NFCIP-1:

- **Activo**: ambos dispositivos generan su propio campo electromagnético, que utilizarán para transmitir los datos.
- **Pasivo**: solo un dispositivo genera el campo electromagnético y el otro se aprovecha de la modulación de la carga para poder transferir los datos. El iniciador de la comunicación es el encargado de generar el campo electromagnético.

El protocolo NFCIP-1 puede funcionar a diversas velocidades como 106, 212, 424 o 848 Kbit/s. Según el entorno en el que se trabaje, las dos partes pueden ponerse de acuerdo de a que velocidad trabajar y reajustar el parámetro en cualquier instante de la comunicación.

3.5.2. Beacons

Según (Kontakt, 2017), los beacons con pequeños radio transmisores Bluetooth que pueden desencadenar acciones en el mundo real o situar de forma precisa una ubicación física por medio de la transmisión de información contextual a dispositivos inteligentes cercanos.

Los beacons se comportan como un faro: transmiten de forma repetida una señal individual que otros equipos pueden ver; la diferencia es que en lugar de emitir luz emiten una señal de radio que consiste en una combinación de letras y números aproximadamente cada décima de segundo. Un dispositivo con Bluetooth, como por ejemplo un smartphone, puede detectar un beacon cuando esté en su radio de acción, de forma parecida a la que los marinos que buscan un faro para saber dónde están.

Fig. 11: Beacon
Imagen tomada de www.infraworld.fr (https://www.infraworld.fr/wp-content/uploads/2015/07/smart-beacon.jpg)

Fig. 12: Desensamblado de un beacon
Imagen tomada de www.mindbowser.com (http://mindbowser.com/wp-content/uploads/2016/05/KontaktIO.png)

Los beacons son dispositivos pequeños y sencillos. Solo constan de una CPU, un emisor de radio y baterías AA.

Los beacons transmiten vía Bluetooth un identificador único que informan a los dispositivos que se encuentran en sus proximidades; por ejemplo, cuando un centro comercial instala beacons en las tiendas, todas ellos tendrán un cierto beacon, con el id registrado en la app que el centro comercial ofrece a sus clientes. Esto significa que un

smartphone puede reconocer inmediatamente que el id de un beacon en concreto es relevante y que pertenece a una determinada tienda.

El ID, en realidad, tiene poco significado; depende de la app reconocerlo y darle contenido y utilizarlo para un propósito determinado. Lo que sucede una vez se haya reconocido el beacon depende de la programación del smartphone; por ejemplo, se puede activar el envío de un cupón de descuento, ofrecer servicios de navegación… Todo lo que debe hacer el beacon es indicar a la app su ubicación exacta, y todo el resto en realidad depende de ella.

Los beacons se pueden organizan en *venues*, que son simplemente grupos de beacons. Normalmente, un venue representa la ubicación física en la que está en grupo de beacons; por ejemplo, se podrían agrupar todos los Beacons de una recepción agrupando todos los beacons situados en ella y llamándolo “Recepción”.

3.6. Datos abiertos de las smart cities

3.6.1. Smart City

Según (Chicano, 2016), las ciudades actuales se están convirtiendo en un entramado de millones de sistemas enredados. En reciente advenimiento y rápida adopción del paradigma de la *smart city* añade nuevas posibilidades vitales, y también ha contribuido significativamente a la intrínseca complejidad de los sistemas de las ciudades.

En una ciudad existen sistemas complejos de modelos de múltiples dominios, como el e-gobierno, gestión del tráfico y del transporte, logística, gestión de la edificación, e-salud… se han convertido en artífices de una innovación sostenida y de una mejora del bienestar de los ciudadanos.

3.6.2. Datos abiertos

Según (Wikipedia-Datos abiertos, 2017), el concepto datos abiertos (open data) es una filosofía y práctica que persigue que determinados tipos de datos estén disponibles de forma libre para todo el mundo, sin restricciones de derechos de autor, de patentes o de otros mecanismos de control.

Tiene una ética similar a otros movimientos y comunidades abiertos, como el software libre, el código abierto (open source, en inglés) y el acceso libre (open access, en inglés).

Son considerados datos abiertos todos aquellos datos accesibles y reutilizables, sin exigencia de permisos específicos.

Los datos abiertos están centrados en material no documental como información geográfica, el genoma, compuestos químicos, fórmulas matemáticas y científicas, datos médicos, biodiversidad, etc. Se trata de fuentes de datos que históricamente han estado bajo el control de organizaciones -públicas o privadas- y cuyo acceso ha estado restringido mediante limitaciones, licencias, copyright y patentes.

3.6.3. Datos abiertos en Madrid, Barcelona y Valencia

Muchas ciudades españolas ponen a disposición del público datos actualizados sobre distintos servicios y sistemas de la ciudad. Por ejemplo, Madrid, Barcelona y Valencia ofrecen en sus respectivos portales de datos abiertos los siguientes conjuntos de datos a fecha 1 de Septiembre de 2017 según (Madrid, 2017), (Barcelona, 2017) y (Valencia, 2017):

Sector	**Madrid**	**Barcelona**	**Valencia**
Ciencia y tecnología	2	2	1
Comercio	5	8	3
Contratación		5	
Cultura y ocio	20	52	5
Demografía	2	110	
Deporte	18	2	
Economía	8		3
Educación	7	1	1
Empleo	2	5	
Hacienda	16		3
Legislación y justicia	1	1	
Medio ambiente	19	13	37
Participación		1	
Recursos humanos		6	
Salud	16		15
Sector público	45	36	5
Seguridad	7	6	1
Sociedad y bienestar	25	9	26
Transporte	50	40	27
Turismo	10	3	10
Urbanismo e infraestructuras	22	43	
Vivienda	1	59	1

Tabla 1: Conjuntos de datos de Madrid, Barcelona y Valencia

4. Propuesta

4.1. Análisis

A continuación se analizará al problema planteado y se establecerán los requisitos que debe cumplir el producto resultante. Para ello, se seguirá la norma IEEE 830, que es un conjunto de buenas prácticas para la especificación de requerimientos del software, y que según (IEEE, 1998) consta de las siguientes partes:

1. Propósito
    1.1. Alcance
    1.2. Definiciones, acrónimos y abreviaturas
    1.3. Referencias
    1.4. Visión general del resto de la especificación
2. Descripción general
    2.1. Perspectiva del producto
    2.2. Funciones del producto
    2.3. Características del usuario
    2.4. Restricciones
    2.5. Suposiciones y dependencias
    2.6. Requisitos futuros
3. Requerimientos específicos

4.1.1. Propósito

Este análisis responde a la necesidad de llevar a cabo un estudio detallado de las acciones que se llevan a cabo desde que un paciente ha sido citado en una clínica hasta que es atendido por el médico, con la intención de obtener una forma de mejorarlas o reducirlas en número mediante el uso de tecnologías de la información. Va dirigido a personas con conocimientos en informática y redes de comunicaciones entre ordenadores, ya que se tratarán conceptos pertenecientes a estas materias cuyo detalle fuera queda del ámbito del presente trabajo.

4.1.1.1. Alcance

El alcance es el siguiente:

- El producto resultante se llamará **Caronte.io**.
- Caronte.io facilitará la citación por medio de transferencia de datos con la clínica y empleando geolocalización y localización de corto alcance.

- El sistema estará enfocado a una clínica situada en el Hospital Casa de Salud de Valencia, no siendo posible ejercer sus funciones sobre más de una clínica.
- El sistema tendrá predefinidas la ubicación geográfica del hospital, no estado contemplado su comportamiento si la clínica cambia de ubicación.
- El sistema requiere que los pacientes dispongan de un smartphone Android con GPS y acceso a Internet, no estando contemplado su funcionamiento en otro tipo de dispositivos.
- El sistema tendrá predefinidos los parámetros de los elementos necesarios para la localización en interiores, no estando contemplado su comportamiento si estos elementos no están disponibles si no funcionan correctamente.
- Queda fuera del alcance de este trabajo el tratamiento del pago por parte del paciente de los servicios prestados.
- Como parte del producto final se contempla las modificaciones necesarias en el sistema de citación de la clínica para integrarlo con Caronte.io.

4.1.1.2. Definiciones, acrónimos y abreviaturas

En los sucesivo utilizaremos los siguientes términos, y son el siguiente significado:

- Utilizaremos el término **App** para referirnos a la parte del sistema que consta de una aplicación para dispositivos móviles.
- Utilizaremos el término **Aplicación** para referirnos a la aplicación de citación de la clínica.
- Utilizaremos el término **Área GPS del Hospital** o simplemente **Área GPS** para referirnos a una superficie comprendida por un círculo con centro en el hospital y de radio de unos cientos de metros. Este radio será definido en Caronte.io y se tomará como una medida de que el paciente se encuentra en los alrededores del hospital.

4.1.1.3. Referencias

Este apartado se ha elaborado bajo las recomendaciones descritas en (IEEE, 1998).

4.1.1.4. Visión general del resto de la especificación

A continuación se hará una descripción del sistema a desarrollar , primero a grandes rasgos, definiendo su función en general y poniéndolo en contexto, y definiendo quienes serán los usuarios potenciales, para después entrar en detalle sobre las condiciones y características que debe cumplir el producto final.

4.1.2. Descripción general

Se desarrollará un sistema de información de naturaleza heterogénea; es decir, bajo un mismo nombre estarán contemplados subproductos tales como una aplicación para smartphones, una aplicación de escritorio, etc.

Los usuarios del sistema serán, por una parte, los pacientes de la clínica, y por otra, los médicos y el personal de recepción.

4.1.2.1. Perspectiva del producto

Caronte.io estará integrado con el sistema de citación y recepción de los pacientes de una clínica.

La aplicación para smartphones contemplada será independiente de otras aplicaciones que pueda tener el paciente instalada en su smartphone.

La App se comunicará con la Aplicación mediante un servicio de paso de mensajes. Estos mensajes se diseñarán de forma adecuada a la funcionalidad que implemente el sistema.

La interfaz de la aplicación del teléfono del paciente debe ser sencilla y sobretodo visual y usable. Se pretende que el sistema sea lo más automático posible, así que se limitarán en lo posible los controles que supongan alguna acción por parte del paciente.

4.1.2.2. Funciones del producto

El sistema soportará las siguientes funciones:

1. La App permitirá comprobar si el paciente tiene cita en la clínica en la fecha en la que se inicie la propia App.

1. La App guiará al paciente en su llegada a la clínica.

2. La App sugerirá al paciente aparcamientos.

3. La App informará del retraso del médico.

4. La Aplicación se comunicará con la App para indicar al paciente que pase a la consulta.

4.1.2.3. Características del usuario

Tal y como se ha dicho en 4.1.1.1. Alcance, los usuarios serán los pacientes de la clínica y y el personal de la misma con atribuciones para gestionar la citación.

No se les supone a ninguno de ellos conocimientos avanzados sobre Tecnologías de la Información, sino tan solo se entiende que los pacientes deben ser capaces de instalar una aplicación en su teléfono móvil y ejecutarla, y que el personal de la clínica conoce su propio sistema de gestión de citas.

4.1.2.4. Restricciones

Las restricciones que se imponen sobre el sistema son las siguientes:

1. La App debe funcionar en dispositivos Android.
2. Debe permitir un mínimo de 100 conexiones concurrentes de smartphones.
3. No debe afectar a ninguna de las actividades de la clínica distintas a aquellas relacionadas con la citación de pacientes.
4. No debe suponer en ningún momento y bajo ninguna circunstancia una interrupción del servicio de la clínica.
5. Debe cumplir con las disposiciones legales vigentes a fecha 10 de Septiembre de 2017 en materia de protección de datos personales.

4.1.2.5. Suposiciones y dependencias

Se harán las siguientes suposiciones:

1. Se supondrá que la Aplicación funciona bajo el sistema operativo Windows 7 Profesional, o Windows 10 Profesional.
2. Se supondrá que la clínica será accesible desde el exterior a través de Internet, mediante una IP fija o algún mecanismo equivalente.

4.1.2.6. Requisitos futuros

Las posibles ampliaciones del sistema se detallan en el apartado 5. Conclusiones. Líneas futuras.

4.1.3. Requerimientos específicos

Dividiremos estos requerimientos en requerimientos funcionales (RF), que son los servicios que proveerá el sistema, y requerimientos no funcionales, que no se refieren directamente a funciones específicas sino a propiedades emergentes de este.

Los requisitos **funcionales** serán los siguientes:

- **RF1**: El sistema contará como mínimo de una aplicación que funcionará en un smartphone Android y de la adaptación necesaria en la aplicación de la clínica.
- **RF2**: El sistema detectará que el paciente está en el Área GPS del hospital, y la Aplicación reflejará dicha circunstancia.
- **RF3**: El sistema facilitará a los pacientes la localización de aparcamientos regulados mediante los datos proporcionados por los portales de Open Data del Ayuntamiento de Valencia.
- **RF4**: El sistema ayudará a los pacientes a localizar la sala de espera.
- **RF5**: El sistema hará innecesario que el paciente anuncie su llegada a la clínica.

- **RF6**: El sistema informará al paciente del posible retraso del médico.
- **RF7**: El sistema permitirá que el personal de la clínica avise al paciente de que debe entrar a la consulta a través de su App.
- **RF8**: La aplicación de citación de la clínica se llama SCOB PU 2017, y fue desarrollada por la empresa In-Fort Servicios Informáticos.

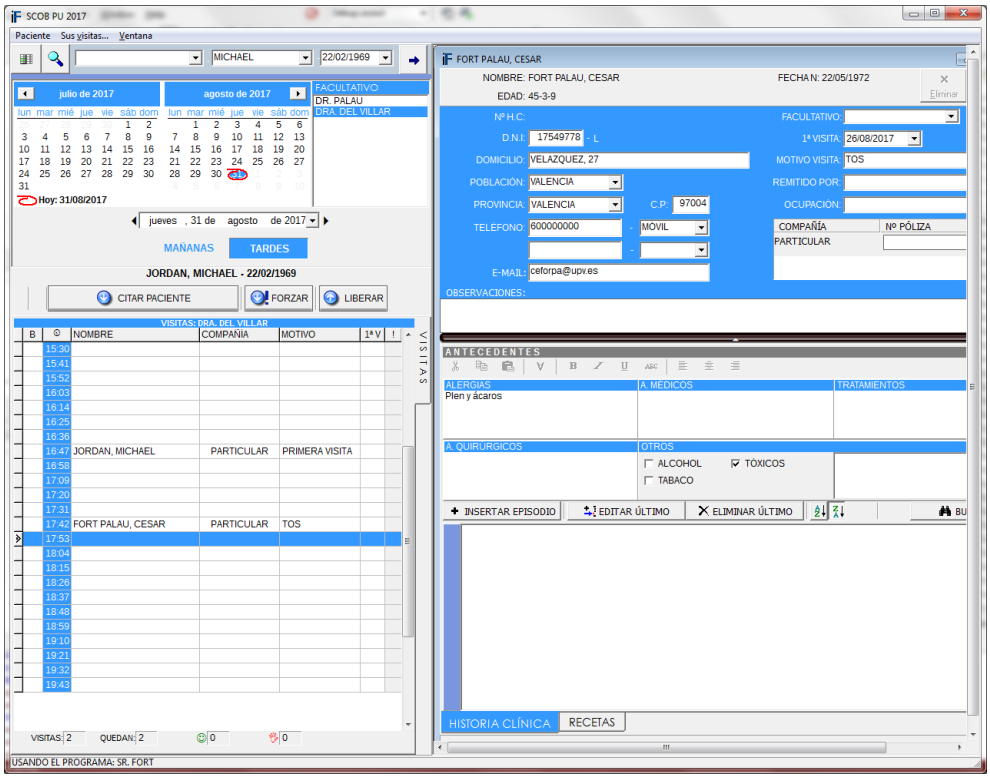


Fig. 13: SCOB PU 2017

Esta aplicación tiene las siguientes características:

- Es una aplicación cliente/servidor realizada en Delphi y que funciona con el servidor de bases de datos MySQL, al que se conecta a través de ODBC.
- Funciona bajo Microsoft Windows 7 y Microsoft Windows 10.
- Dispone de funcionalidades de agenda y de historias clínicas.
- La interfaz tiene una zona de agenda con una rejilla donde aparecen los datos de cada visita, un calendario, y un área donde se ubican los datos de los pacientes, en ventanas independientes.
- De forma cíclica, con una frecuencia personalizable, la aplicación de escritorio actualiza sus datos desde el servidor de bases de datos.
- Identifica a los pacientes de forma unívoca con la terna: [Apellidos], [Nombre], [Fecha de Nacimiento]

- Los datos de filiación de los pacientes se almacenan en la tabla **pacientes**. La descripción de los datos más relevantes de esta tabla es la siguiente:

Tabla: **pacientes**		
Clave primaria	Campo	Tipo
*	Apellidos	varchar(30) NOT NULL
*	Nombre	varchar(20) NOT NULL
*	Fecha_N	date NOT NULL
	N_H_Clinica	varchar(11) NULL
	...	...

- Las visitas de los pacientes se almacenan en la tabla **visitas**. La descripción de los datos más relevantes de esta tabla es la siguiente:

Tabla: **visitas**		
Clave primaria	Campo	Tipo
*	Fecha	date NOT NULL
*	NCol	int(11) NOT NULL
*	Hora	time NOT NULL
	Codigo	varchar(20) NULL
	Apellidos	varchar(30) NULL
	Nombre	varchar(20) NULL
	Fecha_N	date NULL
	Vino	char(1) NULL
	...	...

- El campo **Vino** de la tabla **visitas** contiene la información del estado de la visita, que actualmente se cambia de forma manual. Puede tomar los siguientes valores:
  - S: El paciente está en la sala de espera. En el programa se representa en el *grid* de las visitas con una mano roja: 
  - D: El paciente ha pasado a la consulta. En el programa se representa en el *grid* de las visitas con cara verde sonriente: 

4.2. Diseño

Para llevar a cabo la propuesta, se repasará primeramente el protocolo de llegada y recepción del paciente a la clínica y a partir de ese protocolo modificado y el análisis llevado a cabo en el apartado anterior diseñará el resto del sistema.

4.2.1. Nuevo protocolo de citación

Revisando el diagrama BPMN2 del proceso tradicional y eliminando los elementos que quedan fuera del alcance, esto es, la citación y el pago:

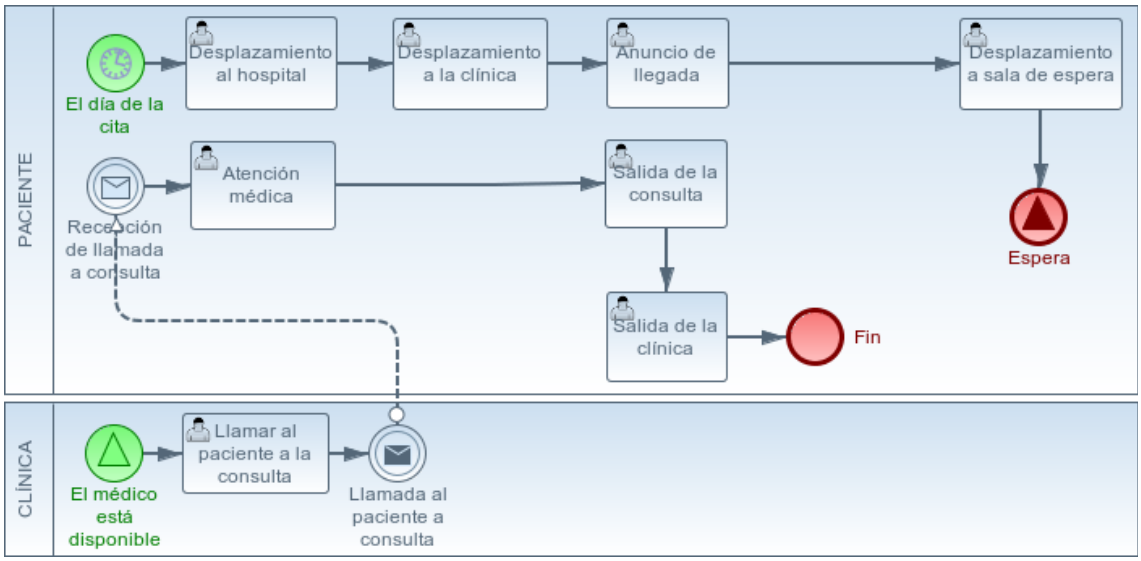


Fig. 14: Diagrama BPMN2 del proceso tradicional simplificado

Según la especificación, el sistema debe hacer innecesario que el paciente anuncie su llegada, con lo que la actividad **Anuncio de llegada** debe pasar a ser automática.

Además, en el diagrama debemos incluir estos elementos obtenidos de la especificación:

- **RF2**: El sistema detectará que el paciente está en el Área GPS del hospital, y la Aplicación reflejará dicha circunstancia.
- **RF3**: El sistema facilitará a los pacientes la localización de aparcamientos regulados mediante los datos proporcionados por los portales de Open Data del ayuntamiento de Valencia.
- **RF4**: El sistema ayudará a los pacientes a localizar la sala de espera.
- **RF5**: El sistema hará innecesario que el paciente anuncie su llegada a la clínica.
- **RF6**: El sistema informará al paciente del posible retraso del médico.

- **RF7**: El sistema permitirá que el personal de la clínica avise al paciente de que debe entrar a la consulta a través de su App

Analizando el diagrama podemos ver que el lugar idóneo para incluir cada uno de los requisitos funcionales serían:

- **RF2**: Como parte de la actividad *Desplazamiento al hospital*
- **RF3**: Como parte de la actividad *Desplazamiento al hospital*
- **RF4**: Como parte de la actividad *Desplazamiento al hospital*
- **RF5**: En lugar de la actividad de usuario *Anuncio de llegada*
- **RF5**: Como parte de la actividad *Desplazamiento al hospital* y *Desplazamiento a la clínica*
- **RF6**: Como parte de la actividad *Llamar al paciente a la consulta*.

Incluyendo estos elementos en el los lugares adecuados del diagrama:

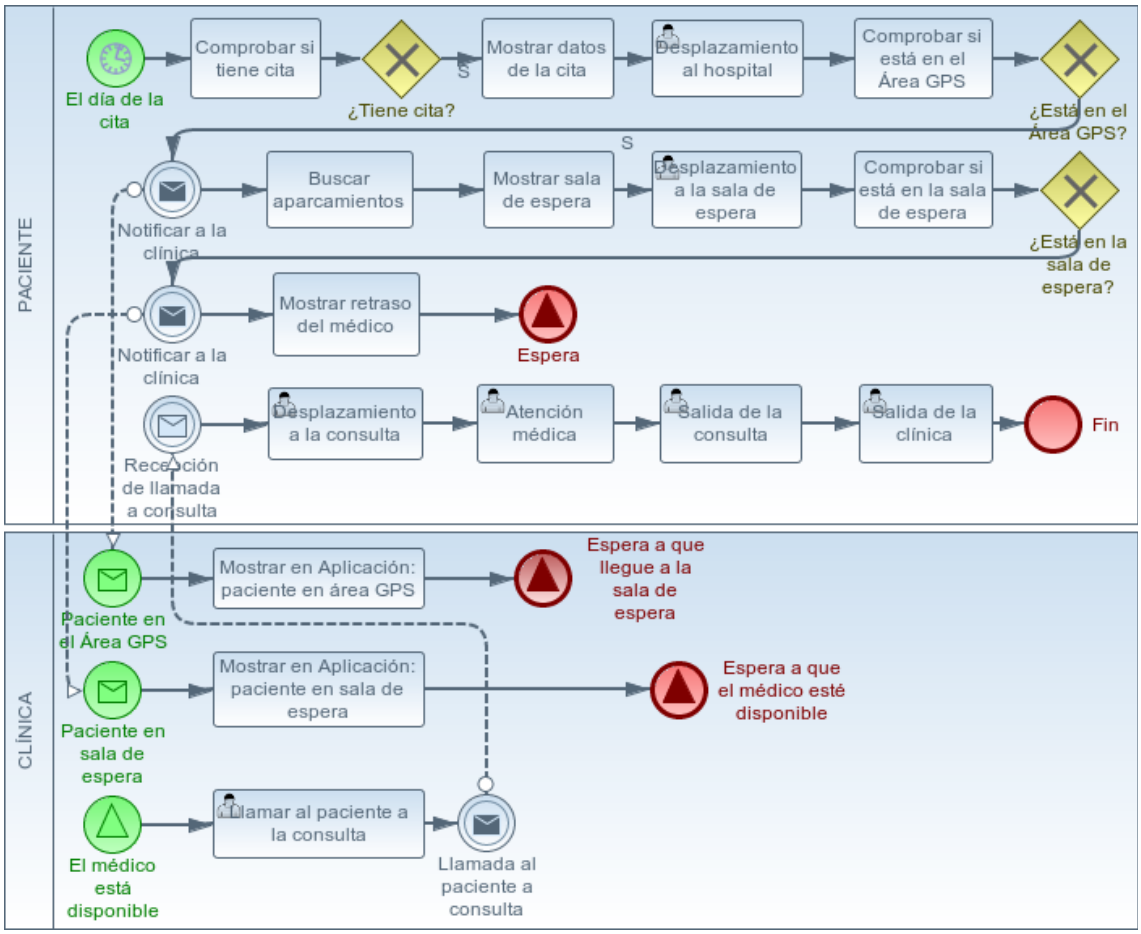


Fig. 15: Diagrama BPMN2 mejorado

Traduciendo en palabras este diagrama:

1. Al iniciar la App, ésta comprobará si el paciente tiene cita ese día.
2. Si la tiene, la App le mostrará los datos de la cita.
3. El paciente se dirigirá a la clínica.
4. Cuando esté cerca de ella, la App dará aviso a la clínica (que mostrará este hecho en la Aplicación), buscará los aparcamientos cercanos y le mostrará a qué sala de espera debe dirigirse cuando llegue al hospital.
5. Cuando llegue a la sala de espera, la App dará aviso a la clínica (que mostrará este hecho en la Aplicación), y mostrará al paciente el retraso actual del médico. El paciente esperará allí a ser llamado a consulta.
6. Cuando el médico esté disponible, la clínica llamará al paciente a la consulta a través de la Aplicación. Esta llamada la recibirá el paciente en su App y pasará a ser atendido.
7. Finalmente, el paciente saldrá de la consulta y abandonará la clínica.

4.2.2. Modelado del sistema

Para modelar este sistema según el estándar UML, recogido en (Booch, 2014), identificaremos a los actores, después construiremos los diagramas de casos de uso, y posteriormente identificaremos las clases que lo componen.

Podemos distinguir tres actores: el paciente, el personal de recepción y el médico, aunque podría darse el caso que estos dos últimos fueran el mismo.

Fig. 16: Actores involucrados

Repasando la Fig. 15: Diagrama BPMN2 mejorado podemos identificar los siguientes casos de uso:

1. Inicio de la App

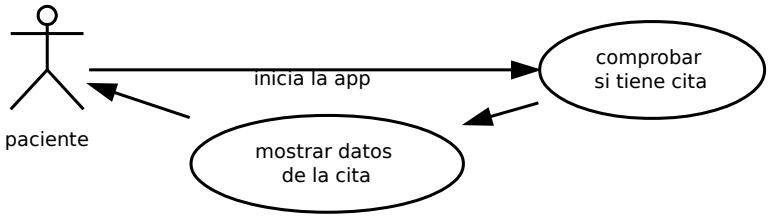


Fig. 17: Caso de uso 1: Inicio de la App

2. El paciente se dirige al hospital:

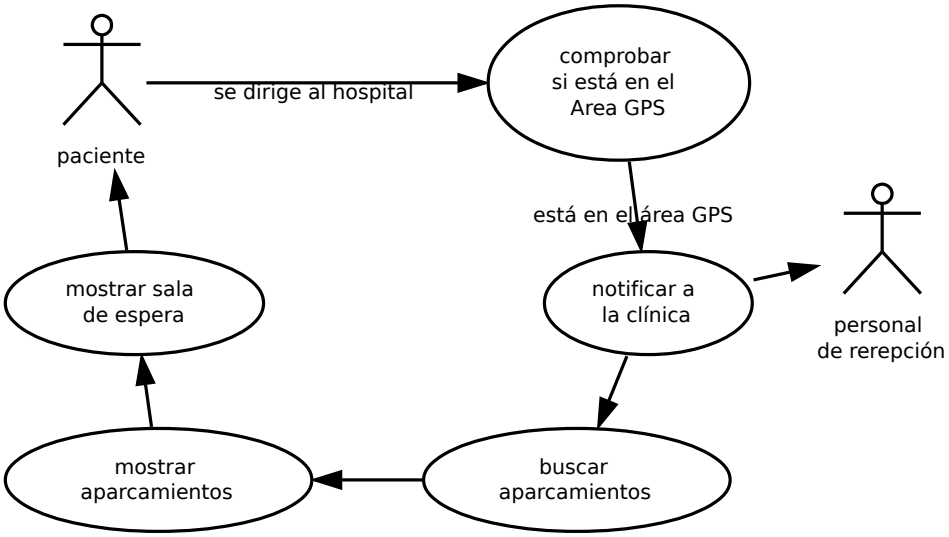


Fig. 18: Caso de uso 2: El paciente se dirige al hospital

3. El paciente va a la sala de espera:

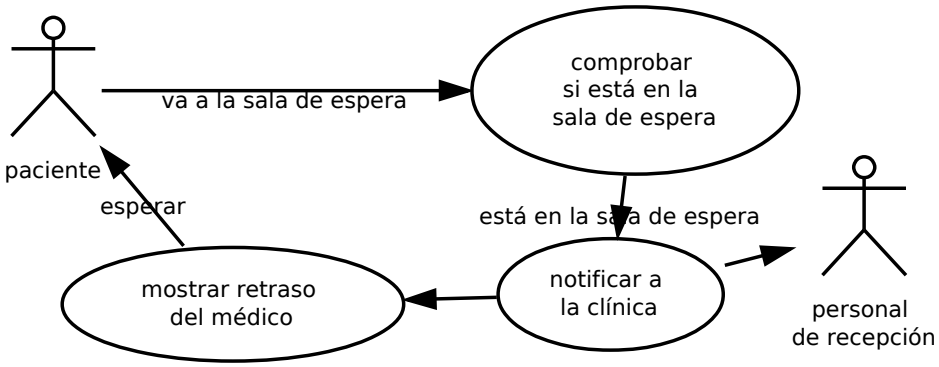


Fig. 19: Caso de uso 3: El paciente va a la sala de espera

4. Se llama al paciente a la consulta:

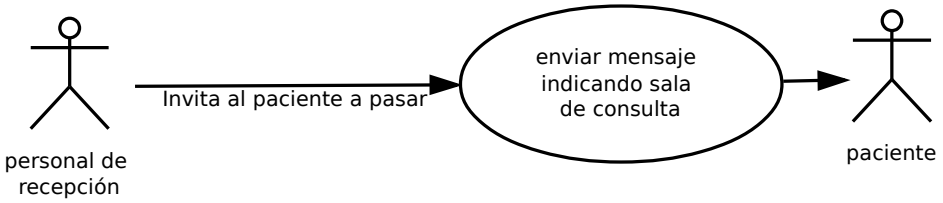


Fig. 20: Caso de uso 4: Llamada al paciente a la consulta

5. Finalmente, el paciente pasa a la consulta, es atendido y abandona el hospital:

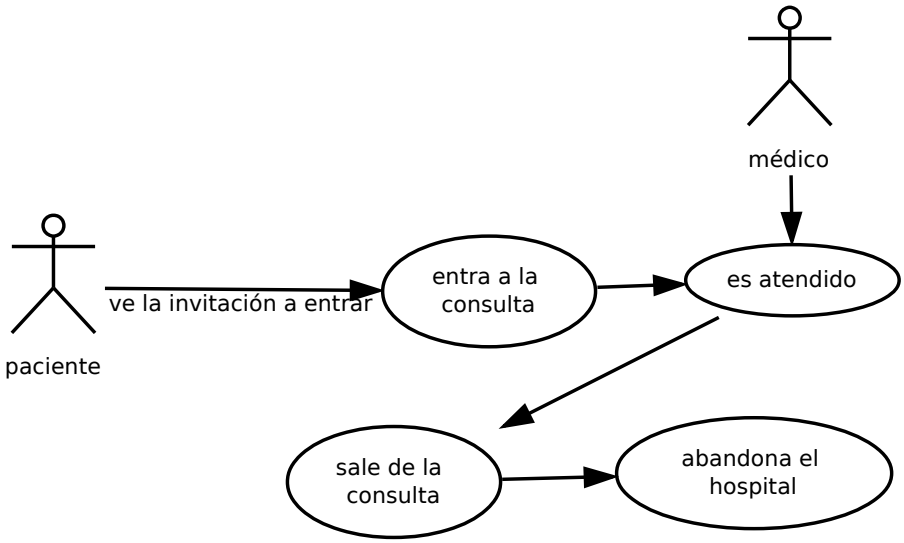


Fig. 21: Caso de uso 5: Llegada a la consulta y atención médica

Asimismo, se pueden identificar las siguientes clases:

1. Clase hospital. Como su nombre indica, representará a los hospitales. Los atributos que consideraremos serán:
    - Nombre
    - Ubicación (coordenadas).
2. Clase aparcamiento. Representará a los aparcamientos regulados. Sus atributos serán:
    - Nombre
    - Dirección
    - Coordenadas GPS

3. Clase clínica. Como su nombre indica, representará a las clínicas. Los atributos que consideraremos serán:
    - Nombre
    - Situación (piso, n.º de consulta)
    - Identificación de su sala de espera
    - Retraso actual en minutos.
4. Clase paciente. Representará a los usuarios de los servicios hospitalarios. Sus atributos serán:
    - Id (de su smartphone)
    - Ubicación (coordenadas)
    - Estado (sin cita, con cita, en el Area del GPS, o en la sala de espera)
5. Clase Sala de Espera. Representará al lugar donde los pacientes permanecen hasta ser atendidos. Su único atributo será su identificador, `Id`.

El diagrama de clases UML es el siguiente:

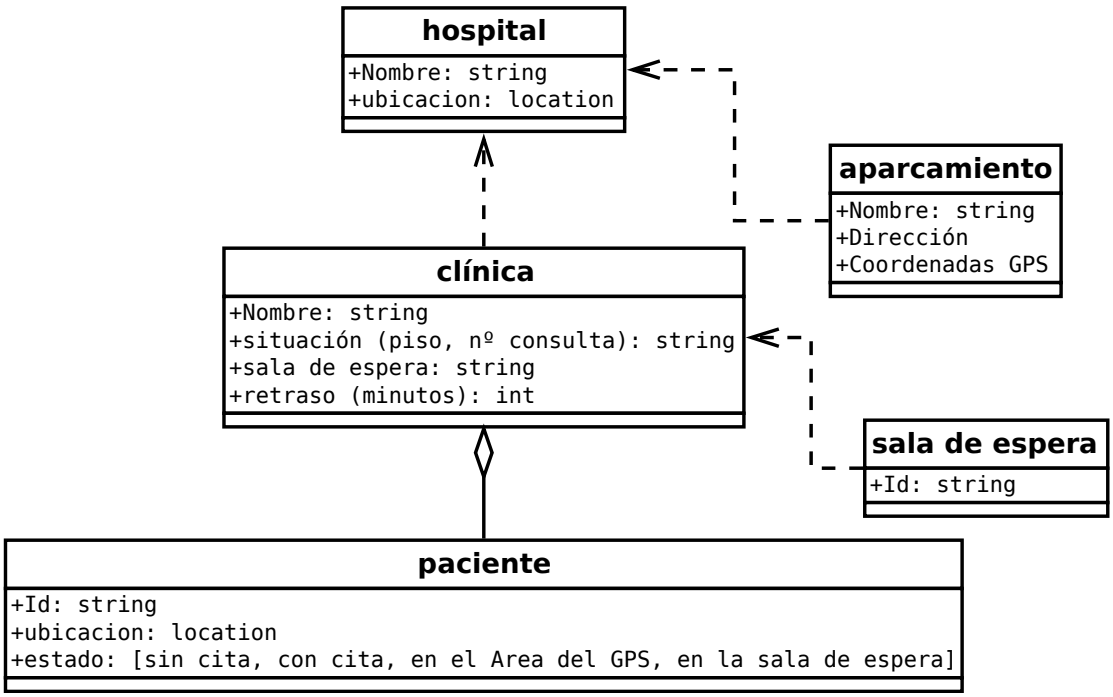


Fig. 22: Diagrama de clases

Las clases tendrán los siguientes métodos:

- Métodos de la clase hospital:
  1. *ObtenerNombre*: Devuelve el nombre del hospital
  2. *ObtenerUbicacion*: Devuelve las coordenadas GPS del hospital
  3. *ObtenerAparcamientos*: Devuelve los aparcamientos cercanos

- Métodos de la clase aparcamiento:
  - *ObtenerDireccion*: Devuelve su dirección
  - *ObtenerCoordenadas*: Devuelve sus coordenadas GPS

- Métodos de la clase clínica:
  - *ObtenerNombre*: Devuelve el nombre de la clínica
  - *ObtenerSituacion*: Devuelve dónde se encuentra la clínica dentro del hospital
  - *ObtenerSalaDeEspera*: Devuelve cual es su sala de espera
  - *LlamarPaciente*: Enviará un mensaje a la App para que el paciente entre a la consulta

- Métodos de la clase paciente:
  1. *ObtenerId*: Devuelve el Id del dispositivo
  2. *ObtenerUbicación*: Devuelve las coordenadas GPS del dispositivo
  3. *Estado*: Devuelve el estado actual
  4. *TengoCita*: Devuelve si el paciente tiene cita
  5. *EstoyEnAreaGPS*: Devuelve si el paciente está en el área GPS del hospital
  6. *NotificarAreaGPS*: Enviará a la clínica un mensaje indicando que el paciente está cerca del hospital
  7. *ObtenerAparcamientos:* Obtendrá los aparcamientos cercanos al hospital
  8. *MostrarAparcamientos*: Mostrará los aparcamientos cercanos al hospital
  9. *ObtenerSalaDeEspera:* Obtendrá la ubicación de la sala de espera de la consulta donde se tiene cita

10. *MostrarSalaDeEspera*: Mostrará la ubicación de la sala de espera de la consulta donde se tiene cita
11. *EstoyEnSalaDeEspera*: DEvolverá si el paciente está en la sala de espera
12. *NotificarSalaDeEspera*: Enviará a la clínica un mensaje indicando que el paciente está en la sala de espera
13. *MostrarCita*: Mostrará por pantalla la información de la cita
14. *MostrarRetraso*: Mostrará el retraso del médico

4.2.3. Arquitectura de sistemas

La especificación del sistema hace necesaria la incorporación de un servicio de intercambio de mensajes. El esquema de la arquitectura de sistemas será será el siguiente:

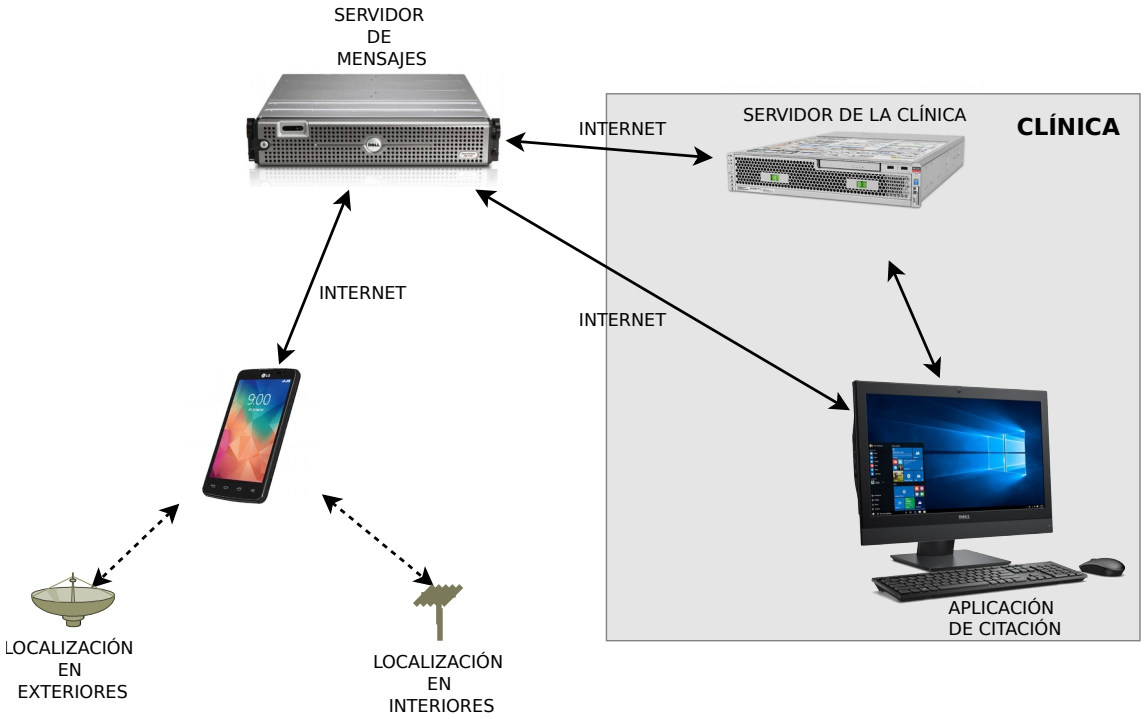


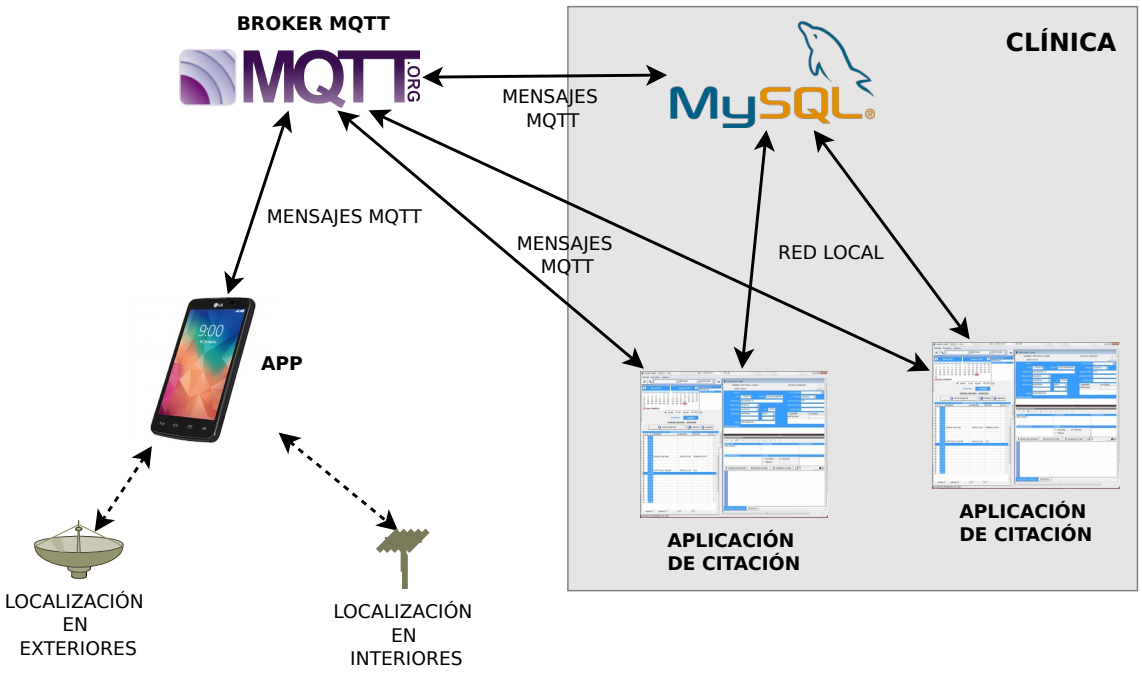
Fig. 23: Arquitectura de sistemas

En este punto, se toma la decisión de utilizar MQTT como protocolo de mensajería porque soporta toda la funcionalidad necesaria manteniendo una gran simplicidad.

Además, hay que tener en cuenta que la aplicación de la clínica, según la especificación de requerimientos:

- Se trata concretamente de la aplicación para Microsoft Windows SCOB PU 2017
- El servidor de bases de datos que gestiona su almacenamiento de información es MySQL

Así pues, la arquitectura de sistemas quedaría de la siguiente forma:



Analizando este diagrama se puede detectar un problema potencial. Repasando los casos de uso, vemos que en los cuatro primeros se accede a la aplicación de gestión para obtener o actualizar datos (que se almacenan en el servidor de bases de datos), y que en hay uso solo de ellos, el que se muestra en la Fig. 20: Caso de uso 4: Llamada al paciente a la consulta, en el que se ejecuta alguna acción que implica comunicación con la App y en el que además el actor encargado de iniciarla es el personal de recepción.

Esta arquitectura provocaría un tráfico excesivo de mensajes MQTT y una sobrecarga de la red, con lo que no es aceptable.

Dado que el servidor MySQL gestiona los datos de la aplicación, se puede implementar una aplicación intermedia o *middleware* que se encargue de gestionar las peticiones desde el servidor MQTT. Esta aplicación estaría suscrita a las colas MQTT adecuadas y haría las lecturas o escrituras necesarias en la base de datos.

El dejarlo así provoca un nuevo problema, y es que las llamadas al paciente no serían en tiempo real, sino que habría que esperar a la sincronización de la base de datos para que al paciente le llegara el mensaje de entrar, con lo que pasarían unos segundos que no harían operativo el sistema y sería rechazado por falta de usabilidad.

Como consecuencia, lo que finalmente se propone es que la aplicación se conecte directamente al servidor MQTT, pero exclusivamente para implementar la funcionalidad de llamar al paciente a la consulta, o sea siempre **publicando** mensajes.

De este modo, la arquitectura de sistemas definitiva es la siguiente:

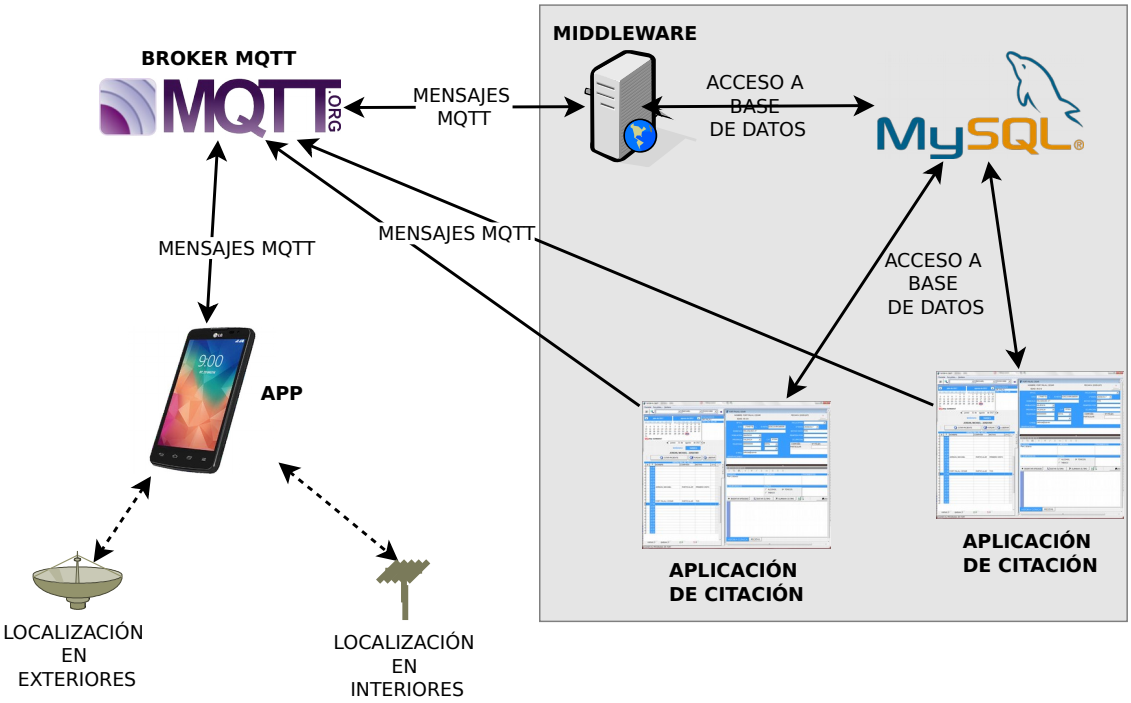


Fig. 24: Arquitectura de sistemas definitiva

Una vez establecida la arquitectura es el momento de diseñar los tres elementos que van a componer finalmente el sistema, pero es necesario diseñar el protocolo del paso de mensajes y la localización el interiores para que los diseños sean coherentes, así que el diseño se llevará a cabo en el siguiente orden:

1. Se diseñarán las modificaciones a llevar a cabo en la aplicación de la clínica. Ello implicará diseñar qué mensajes manejará el servidor MQTT

1. Se hará un diseño inicial de la App

2. Se diseñará el protocolo de paso de mensajes

3. Se diseñará la localización en interiores

4. Se completará el diseño de la App

5. Se diseñará el *middleware*

4.2.4. Diseño de las modificaciones en la aplicación de la clínica

Los cambios que será necesario llevar a cabo en la aplicación de la clínica son los siguientes:

1. Para asociar al paciente de forma unívoca con su teléfono móvil se añadirá a los datos de filiación de los pacientes el campo `Id Movil`, que contendrá el identificador único del teléfono o el número de serie del smartphone del paciente.

2. Instalar una librería o componente que permita la comunicación con un servidor MQTT.

3. Según la especificación, la aplicación debe de mostrar que el paciente se encuentra en el Área GPS en cuanto ello suceda. Para ello, a los estados posibles del paciente que se detallan en el apartado 4.1.3. Requerimientos específicos se le añadirá un nuevo estado que llamaremos G y que se representará con el icono de un edificio  .

4. Según la especificación, la aplicación debe de mostrar que el paciente se encuentra en la sala de espera de forma automática sin anunciar su visita. El estado que indicaba esta circunstancia ya existía (), así que no será necesario añadir ningún estado nuevo para reflejar esta circunstancia.

1. Según la especificación, se debe extender la funcionalidad de la aplicación para indicar que un paciente está en la consulta para que además se envíe un mensaje a la App del paciente invitándole a entrar.

4.2.5. Diseño previo de la App

En el el apartado 4.2.2. Modelado del sistema vimos que la clase paciente, que se implementa por medio de la App, debe implementar los siguientes métodos:

1. *ObtenerId*
2. *ObtenerUbicación*
3. *Estado*
4. *TengoCita*
5. *EstoyEnAreaGPS*
6. *NotificarAreaGPS*
7. *ObtenerAparcamientos*
8. *MostrarAparcamientos*
9. *ObtenerSalaDeEspera*
10. *MostrarSalaDeEspera*
11. *EstoyEnSalaDeEspera*
12. *NotificarSalaDeEspera*
13. *MostrarCita*
14. *MostrarRetraso*

En detalle, las acciones que debe llevar a cabo cada método son las siguientes:

1. ***Obtener Id***. La App utilizará las funciones adecuadas del sistema operativo para obtener el número de serie del teléfono o equivalente. Este valor estará visible en la interfaz para que el paciente lo pueda comunicar a la clínica y así sea introducido en el campo `Id Movil` de su aplicación.

2. ***ObtenerUbicacion***. La App utilizará las librerías de geolocalización adecuadas para obtener su ubicación o, por lo menos, su última ubicación conocida.

3. ***Estado***. Analizando la Fig. 15: Diagrama BPMN2 mejorado podemos distinguir 5 estados:

   - En espera. El paciente no tiene cita
   - Paciente con cita. El paciente tiene una cita planificada
   - Dentro del Área GPS. El paciente está en las inmediaciones del hospital
   - En la sala de espera. El paciente ha llegado a la sala de espera
   - Dentro de la consulta. El paciente está siendo atendido por el médico.
   - Estos estados se numerarán del 0 al 4. En el siguiente diagrama se muestran los estados junto con los eventos que provocan los cambios de estado:

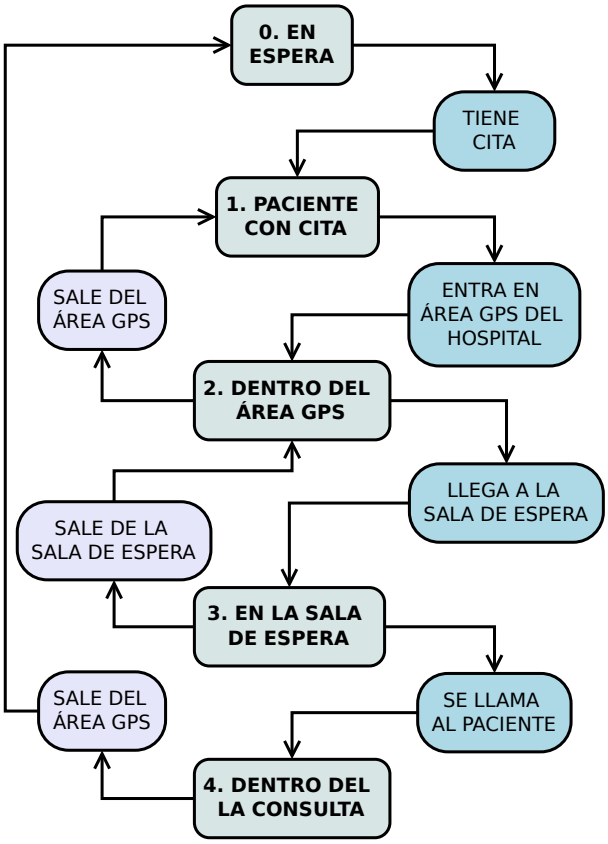


Ilustración 1: Diagrama de estados de la App

4. ***TengoCita***. La App publicará un mensaje en el servidor MQTT para preguntar si tiene cita. Si es así, el sistema le responderá con un mensaje en una cola MQTT a la que estará suscrita la App indicándole los detalles de la cita.

5. ***EstoyEnAreaGPS***. La App calculará la distancia entre su posición y la ubicación del hospital, y si es menor que una cantidad predefinida devolverá TRUE , y en caso contrario FALSE.

6. ***NotificarAreaGPS***. En caso de que el valor devuelvo por EstoyEnAreaGPS sea TRUE, la App publicará un mensaje en una de las colas del servidor MQTT indicando esta circunstancia.
7. ***ObtenerAparcamientos***. La App obtendrá mediante el intercambio de mensajes los aparcamientos cercanos al hospital.
8. ***MostrarAparcamientos***. La App mostrará los aparcamientos cercanos al hospital.
9. ***ObtenerSalaDeEspera***. La App mostrará dónde se encuentra la sala de espera a la que debe dirigirse el paciente.
10. ***MostrarSalaDeEspera***. La App mostrará dónde se encuentra la sala de espera a la que debe dirigirse el paciente.
11. ***EstoyEnSalaDeEspera***. Devolverá TRUE si el paciente está en la sala de espera y FALSE en caso contrario.
12. ***NotificarSalaDeEspera***. En caso de que el valor de EstoySalaDeEspera sea TRUE, la App publicará un mensaje en una de las colas del servidor MQTT indicando esta circunstancia.
13. ***MostrarCita***. La App mostrará los detalles de la cita
14. ***MostrarRetraso***. La App mostrará por pantalla el retraso del médico

4.2.6. Diseño del protocolo de paso de mensajes

En este punto deberemos definir cómo será la suscripción y publicación de mensajes. En la App hemos visto que necesitarán este servicio los métodos:

- *TengoCita*
- *NotificarAreaGPS*
- *ObtenerAparcamientos*
- *ObtenerSalaDeEspera*
- *NotificarSalaDeEspera*

Así pues, se definirán las siguientes colas MQTT:

Cola	**Descripción**	**Atributos**	**Publicará/ Estará Suscrito**
Aparcamientos	Dirección de los aparcamientos cercanos al hospital	• Aparcamientos cercanos	P: Aplicación S: App
DispositivosAparcamiento	Consulta de los aparcamientos	• Id dispositivo	P: App S: Aplicación

Cola	**Descripción**	**Atributos**	**Publicará/ Estará Suscrito**
	cercanos		
DispositivosEspera	Tiempos de espera del médico	• Id dispositivo • Espera (min)	P: Aplicación S: App
DispositivosCita	Dispositivos con cita en el día	• Id dispositivo • Hora de la cita • Lugar de la cita	P: Aplicación S: App
DispositivosGPS	Dispositivos en el Área GPS	• Id dispositivo	P: App S: Aplicación
DispositivosSEspera	Dispositivos en la sala de espera	• Id dispositivo	P: App S: Aplicación
EntrarPaciente	Indica al paciente que entre	• Id dispositivo • Lugar de consulta	P: Aplicación S: App
MensajesAplicacion	Mensajes desde la aplicación a los dispositivos	• Id dispositivo • Sala de Espera	P: Aplicación S: App
TengoCita	Consulta de la app de si tiene cita ese día	• Id dispositivo	P: App S: Aplicación

Tabla 2: Descripción de las colas MQTT

Los formatos de los mensajes serán los siguientes:

Cola	**Formato Del Mensaje**
Aparcamientos	[Texto]
DispositivosAparca-miento	[Id del dispositivo]
DispositivosSEspera	1=[Id del dispositivo], si el dispositivo ha entrado en la sala de espera 0=[Id del dispositivo], si el dispositivo ha salido de la sala de espera
DispositivosCita	[Id del dispositivo]=[Lugar de la cita]%[Hora de la cita]
DispositivosGPS	1=[Id del dispositivo], si el dispositivo ha entrado en el Área GPS 0=[Id del dispositivo], si el dispositivo ha salido del Área GPS
EntrarPaciente	[Id del dispositivo]=[Lugar de la consulta]
MensajesAplicacion	[Id del dispositivo]=[Línea 1]%[Línea 2]
TengoCita	[Id del dispositivo]

Tabla 3: Formato de los mensajes

De forma gráfica:

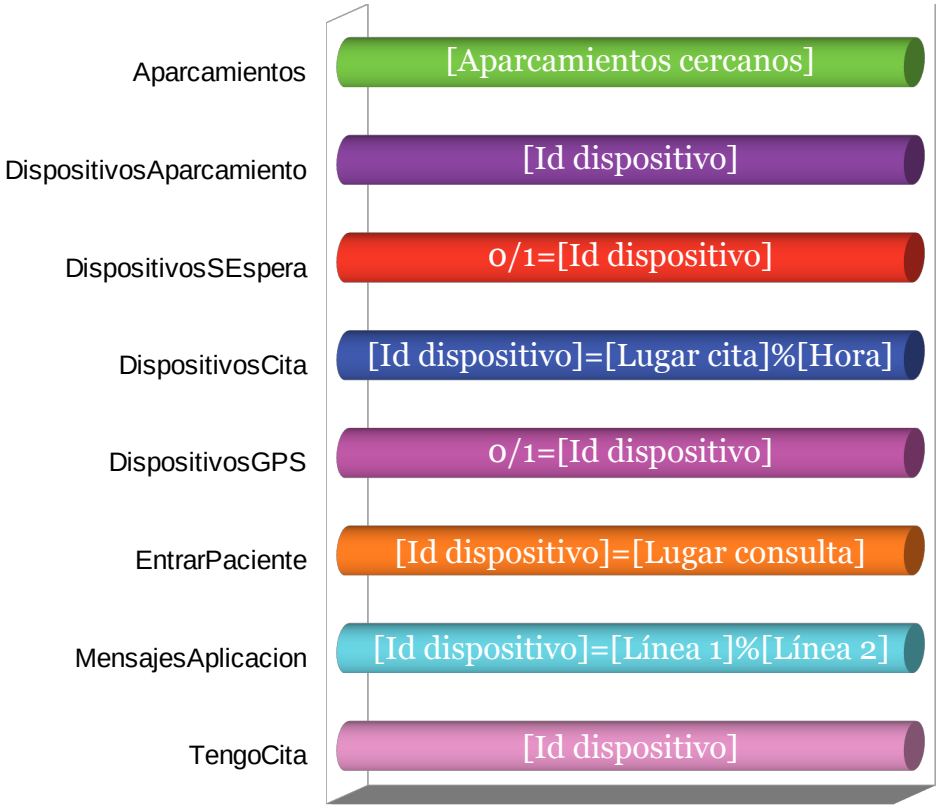


Fig. 25: Colas MQTT

4.2.7. Diseño de la localización en interiores

Para la localización en interiores se utilizarán los beacons del fabricante Kontakt.io. En principio solo es necesario detectar la proximidad a una sala de espera, así para efectos del presente proyecto se utilizará solo uno de ellos, que estará ubicado allí.

Los beacons se comunican por Bluetooth, cuyo alcance es de alrededor de 10 metros. De este modo, cuando la App detecte el beacon en las proximidades asumirá que el paciente está en las cercanías de la sala de espera.

En este proyecto se han utilizado unos beacons Eddystone de Kontakt.io.

Fig. 26: Beacon. Vista superior

Fig. 27: Beacon. Vista trasera

Fig. 28: Beacons Eddystone Kontakt.io

4.2.8. Diseño completo de la App

Conociendo las colas MQTT ya es posible completar el diseño de la App.

La App, que implementa la clase paciente, tendrá finalmente los siguientes métodos:

1. **ObtenerIdAndroid**. Obtendrá el Id del dispositivo y se corresponderá con el método *ObtenerId* del modelo del sistema.

2. **ObtenerUbicación**. Obtendrá la ubicación del dispositivo, o más exactamente la última ubicación conocida, y calculará la distancia al centro médico. Se corresponderá con el método *ObtenerUbicación* del modelo.

3. **ActualizarEstado**. Gestionará el cambio entre estados y se corresponderá con el método *Estado* del modelo del sistema.

4. **ObtenerAreaGPS**. Será un proceso que cada 20 segundos comprobará si el dispositivo está en un radio de 400 metros del centro médico. Se corresponderá con los métodos *EstoyEnAreaGPS* y *NotificarAreaGPS* del modelo.

5. **BuscarBeacons**. Localizará los beacons que se encuentren en las inmediaciones. Implementará los métodos *MostrarSalaDeEspera* y *EstoyEnSalaDeEspera* del modelo.

6. **Suscribirse**. Permitirá suscribir a la app a las colas MQTT, que como se ha visto anteriormente son: Aparcamientos, DispositivosCita, EntrarPaciente y MensajesAplicacion.

Además serán necesarios los siguientes métodos para comunicarse con el servidor MQTT:

1. **IniciarMQTT**. Iniciará un proceso que permanecerá a la escucha de los mensajes procedentes de MQTT.
    - Si llega un mensaje procedente de la cola Aparcamientos, mostrará por pantalla la lista de aparcamientos contenida en el mensaje.
    - Si llega un mensaje procedente de DispositivosCita:
        1. Descompondrá el mensaje para identificar si el Id corresponde con el suyo
        2. Si es así, así mostrará por pantalla el mensaje Tiene cita en X a las Y, siendo X e Y el centro hospitalario e y la hora contenidos en el mensaje respectivamente y llamará a ActualizarEstado para pasar al estado 1.
    - Si llega un mensaje procedente de EntrarPaciente:
        1. Descompondrá el mensaje para identificar si el Id corresponde con el suyo.
        2. Si es así, mostrará por pantalla en mensaje "Por favor, pase a la consulta:", junto con el nombre de la consulta contenido en el segundo fragmento del mensaje y llamará a ActualizarEstado para pasar al estado 4.
    - Si llega un mensaje procedente de MensajesAplicacion:
        1. Descompondrá el mensaje para identificar si el Id corresponde con el suyo
        2. Si es así, mostrará en pantalla los otros dos fragmentos del mensaje

2. **PublicarMensaje**. Permitirá publicar mensajes en MQTT. Se elegirá en todos los casos la estrategia *por lo menos una vez* descrita en el apartado 3.3.3. MQTT. MQTT sobre WebSockets.

4.2.9. Diseño del *middleware*

Una vez con las colas MQTT ya definidas podemos acometer el diseño del *middleware*. Este módulo implementará las clases *hospital*, *aparcamiento* y *clínica* del modelo.

Para gestionar la búsqueda de aparcamientos de una forma sencilla se creará en la base de datos una tabla llamada `aparcamientos`, donde se importarán los datos descargados del portal de Open Data del ayuntamiento de Valencia relativos a aparcamientos regulados. Esta tabla tendrá la siguiente estructura:

Tabla: **aparcamientos**		
Clave primaria	Campo	Tipo
*	Id	int(11) NOT NULL
	x	double NULL
	y	double NULL
	nombre	varchar(254) NULL
	direccion	varchar(254) NULL

El módulo intermedio tendrá los siguientes cometidos:

1. Estará suscrito a las colas DispositivosGPS, DispositivosSEspera, TengoCita y DispositivosAparcamiento

2. Cuando reciba un mensaje en la cola DispositivosSEsper:
   - Descompondrá el mensaje para obtener el Id del dispositivo
     - Si el comando es 1:
       1. Actualizará en la tabla visitas el campo Vino al valor S
       2. Obtendrá el retraso actual del médico
       3. Formateará un mensaje de la forma: ”Espere aquí, por favor” %”Retraso:”[retraso]
       4. Publicará en la cola MensajesAplicacion el mensaje anterior
     - Si el comando es 0, dado que un paciente que sale del área del beacon necesariamente estará en el área GPS, actualizará en la tabla visitas el campo Vino al valor G.

3. Cuando reciba un mensaje en la cola DispositivosGPS:
   - Descompondrá el mensaje en: comando (1 ó 0) e Id del dispositivo
     - Si el comando es 1:
       5. actualizará en la tabla visitas el campo Vino al valor G
       6. Obtendrá el retraso actual del médico
       7. Formateará un mensaje de la forma: [id dispositivo]=”Diríjase a: “[Sala de espera]%”Retraso: “[retraso];
       8. Publicará en la cola MensajesAplicacion el mensaje anterior

- Si el comando es 0, actualizará en la tabla visitas el campo Vino al valor N

3. Cuando reciba un mensaje en la cola TengoCita:
    1. Descompondrá el mensaje para obtener el Id del dispositivo
    2. Buscará en la base de datos si el paciente tiene cita ese día
        - Si tiene cita
            1. Compondrá un mensaje de la forma: [idPaciente]=[Nombre de la clínica]%[Hora de la cita]
            2. Enviará por la cola DispositivosCita el mensaje anterior
4. Cuando reciba un mensaje en la cola DispositivosAparcamiento
    1. Obtendrá los aparcamientos cercanos
    2. Compondrá un mensaje con la lista de los aparcamientos obtenidos
    3. Publicará el mensaje en la cola Aparcamientos

4.3. Implementación

La implementación de Caronte.io conlleva la implementación de todos y cada uno de sus 4 elementos:

1. Instalación y configuración del broker MQTT Mosquitto.
2. Implementación de la app.
3. Implementación del middleware.
4. Implementación de los cambios en la aplicación Ía clínica.

Se toman las siguientes decisiones:

1. Como broker MQTT, después de valorar en profundidad la plataforma Amazon AWS IoT se opta por Mosquitto por su simplicidad, ligereza y facilidad de instalación y configuración.
2. Se utilizará MQTT sobre WebSockets para potenciar la escalabilidad del sistema.
3. Por simplicidad no se utilizarán mecanismos de seguridad: autentificación y cifrado de los mensajes, pero estos aspectos deberán ser tenidos en cuenta en la implantación en un entorno de producción.

4.3.1. Instalación y configuración del broker MQTT Mosquitto

Para llevar a cabo el desarrollo y la implementación se instala en la red local de la clínica un servidor dedicado con el sistema operativo Fedora 25.

El servidor Mosquitto se puede instalar fácilmente desde el repositorio del propio sistema operativo. Una vez instalado es necesario modificar el archivo `etc/mosquitto/mosquitto.conf` y añadir el servicio a los scripts de inicio del sistema. Las opciones configuradas en `mosquitto.conf` se pueden ver en el Anexo I. Configuración de Mosquitto.

Una vez instalado Mosquitto, para tener acceso desde el exterior es necesario redirigir en el router de la clínica el puerto 1883 mediante NAT a la ip interna del servidor, de forma que las conexiones MQTT entrantes desde el exterior sean gestionadas por éste.

4.3.2. Implementación de la app

La aplicación se ha desarrollado en Android Studio. Consta de una única *activity*, en la que se van mostrando u ocultando componentes según sea necesario.

Se han utilizado las siguientes librerías y componentes:

- Para gestionar la geolocalización se ha optado por el paquete Location de la API de Google para Android, documentada en (Google-location, 2017).
- Para gestionar la comunicación a través del protocolo MQTT se ha optado por el paquete Eclipse Paho, cuya documentación está disponible en disponibles en (Eclipse-paho, 2017). Esta comunicación implementa mediante un servicio.
- Para gestionar las comunicaciones con los beacons se ha utilizado el API proporcionado por Kontakt.io, cuya documentación está disponible en: (Kontakt-Developer, 2017). Esta comunicación también se ha implementado por medio de un servicio.

Es importante reseñar que la app necesita que se registren adecuadamente en el fichero **`AndroidManifest.xml`** los permisos y los servicios que se van a utilizar, así como registrar una API KEY que previamente se habrá obtenido en el portal de desarrolladores de Kontakt.io:

```
...
    <uses-permission android:name="android.permission.INTERNET" />
                                            <uses-permission
      android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission
      android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission
      android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN"
      />
```

```
...
        <service
            android:name="org.eclipse.paho.android.service.MqttService"
            android:exported="false" />
        <service
           android:name =
             "com.kontakt.sdk.android.ble.service.ProximityService"
            android:exported="false" />
        <service
            android:name="service.BackgroundScanService"
            android:enabled="true"
            android:exported="false" />
        <meta-data
            android:name="kontakt.io.API_KEY"
            android:value="@string/kontakt_io_api_key" />
...
```

Detalle de AndroidManifest.xml

Para representar los cinco estados de la app se ha diseñado, apoyándose en las recomendaciones sobre el diseño de interfaces de usuario recogidas en (Shneiderman, 2009) la siguiente representación de los mismos:

Fig. 29: Estado 0. En espera

Fig. 30: Estado 1. Con cita

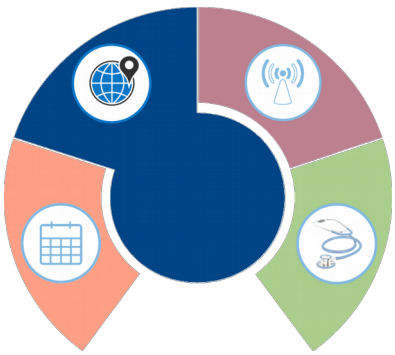


Fig. 31: Estado 2. En el Área GPS

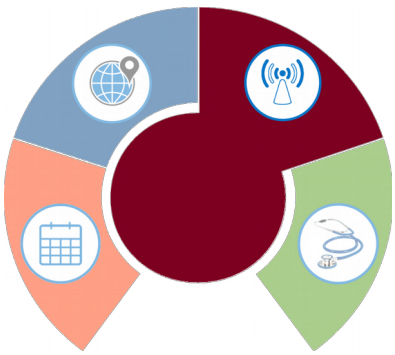


Fig. 32: Estado 3. En la sala de espera

Fig. 33: Estado 4. En la consulta

A estas representaciones de los estados se le añade un botón de inicio y un par de cajas de texto para mostrar la ip del servidor y el identificador del smartphone, que el paciente comunicará al personal de recepción de la clínica, y así conformar la interfaz final:

Fig. 34: Interfaz de la App

El cambio de estados se lleva a cabo mediante el método **`actualizarEstado(int estadoActual, int nuevoEstado)`**, que según el estado actual y el nuevo lleva a cabo las acciones oportunas:

```
protected void actualizarEstado(int estadoActual, int nuevoEstado) {
    // Estado = 0 : En espera
    // Estado = 1 : Con cita
    // Estado = 2 : Dentro del área de GPS
    // Estado = 3 : En el área del Beacon
    // Estado = 4 : En la consulta
    String idDispositivo=id;
    if (nuevoEstado == 1) {
        // Estado = 1 : Con cita
        estado=1;
        imagen0.setVisibility(View.GONE);
        ...
        aparcamientos.setVisibility(View.GONE);
        getLastLocation();
        if (estaCerca) {
            actualizarEstado(estado,2);
        }
        if (estadoActual==2) {    // Sale del área del gps
            publish("DispositivosGPS", "0=".concat(idDispositivo));
        }
    } else if (nuevoEstado == 2) {
        // Estado = 2 : Dentro del área de GPS
        estado=2;
        imagen0.setVisibility(View.GONE);
        ...
        // Notifica que está cerca
        idDispositivo
       android.provider.Settings.System.getString(getContentResolver(),
       android.provider.Settings.System.ANDROID_ID);
        publish("DispositivosGPS", "1=".concat(idDispositivo));
        // Muestra aparcamientos
        publish("DispositivosAparcamiento", idDispositivo);
        // Inicia la búsqueda de beacons
        startBackgroundService();
        if (estadoActual==3) {    // Sale del área del beacon
            publish("DispositivosSEspera", "0=".concat(idDispositivo));
        }
    } else if (nuevoEstado == 3) {
        // Estado = 3 : En la sala de espera
        estado=3;
        imagen0.setVisibility(View.GONE);
        ...
        // Anuncia que ha llegado
        idDispositivo=
      android.provider.Settings.System.getString(getContentResolver(),
      android.provider.Settings.System.ANDROID_ID);
        publish("DispositivosSEspera", "1=".concat(idDispositivo));
    } else if (nuevoEstado == 4) {
        // Estado = 4 : En la consulta
        estado=4;
        stopBackgroundService(); // Deja de buscar beacons
        imagen0.setVisibility(View.GONE);
```

```
...
        aparcamientos.setVisibility(View.GONE);
    } else {
        // Estado = 0 : En espera
        estado=0;
        imagen0.setVisibility(View.VISIBLE);
        ...
        mensajes.setText("Bienvenido a CARONTE.io");
        mensajes1.setText("");
    }
}
```

Para obtener la ubicación se ha implementado el método **`obtenerUbicacion()`**, que obtiene la ubicación actual y la compara con la del hospital y hace llamadas al método **`getLastLocation()`**:

```
public void obtenerUbicacion() {
    try {
       mFusedLocationClient.getLastLocation().addOnSuccessListener(this,
       new OnSuccessListener<Location>() {
            @Override
            public void onSuccess(Location location) {
                if (location != null) {
                    float distancia=0;
                    Location ubicacionHospital=new
                    Location("hospital");
                    ubicacionHospital.setLongitude(longitudHospital);
                    ubicacionHospital.setLatitude(latitudHospital);
                    distancia = location.distanceTo(ubicacionHospital);
                    Toast.makeText(MainActivity.this, "Distancia al
                    hospital...".concat(Float.toString(distancia)),
                    Toast.LENGTH_SHORT).show();
                    if (distancia < radioMetros) {
                        estaCerca = true;
                        if (estado==1) //Si tiene cita
                            actualizarEstado(estado,2);
                    } else {
                        if (estado==4) //HA sido visto y se va del
                          hospital
                            actualizarEstado(estado,0);
                        estaCerca = false;
                    }
                }
            }
        });
    }
    catch (SecurityException e) {
        System.out.println("Error " + e.getMessage());
    }
}

private void getLastLocation() {
```

```
      mFusedLocationClient.getLastLocation().addOnCompleteListener(this,
new OnCompleteListener<Location>() {
                @Override
                public void onComplete(@NonNull Task<Location> task) {
                    float distancia=0;
                        if (task.isSuccessful() && task.getResult() !=
                    null) {
                        mLastLocation = task.getResult();
                        latitud = String.format("%f",
                           mLastLocation.getLatitude());
                        longitud = String.format("%f",
                           mLastLocation.getLongitude());
                        Location ubicacionHospital=new
                           Location("hospital");

                    ubicacionHospital.setLongitude(longitudHospital);
                        ubicacionHospital.setLatitude(latitudHospital);
                        distancia =
                           mLastLocation.distanceTo(ubicacionHospital);
                        if (distancia < radioMetros) {
                            estaCerca = true;
                            actualizarEstado(estado,2);
                        } else {
                            estaCerca = false;
                            actualizarEstado(0,1);
                        }
                    } else {
                        Log.w(TAG, "getLastLocation:exception",
                           task.getException());
                        dataReceived.setText("UBICACION NO
                           ENCONTRADA");
                    }
                }
            });
}
```

Para gestionar la conexión MQTT se crea el servicio **MqttHelper**:

```
package helpers;
import android.content.Context;
...
public class MqttHelper {
    public MqttAndroidClient mqttAndroidClient;
    final String serverUri = "ws://83.48.35.198:1883";
    final String clientId = "CaronteAplicacion";
    final String username = "";
    final String password = "";
    public MqttHelper(Context context){
        mqttAndroidClient = new MqttAndroidClient(context, serverUri,
       clientId);
        mqttAndroidClient.setCallback(new MqttCallbackExtended() {
            @Override
            public void messageArrived(String topic, MqttMessage
             mqttMessage) throws Exception {
```

```
                Log.w("Mqtt", mqttMessage.toString());
            }
            @Override
            public void deliveryComplete(IMqttDeliveryToken
             iMqttDeliveryToken) {
            }
        });
        try {
            connect();
            }
        catch (Exception e) {
                System.out.println("Error " + e.getMessage());
            }
    }
    public void setCallback(MqttCallbackExtended callback) {
        mqttAndroidClient.setCallback(callback);
    }
    private void connect() throws MqttException {
        MqttConnectOptions mqttConnectOptions = new
         MqttConnectOptions();
        mqttConnectOptions.setAutomaticReconnect(true);
        mqttConnectOptions.setCleanSession(false);
        try {
            mqttAndroidClient.connect(mqttConnectOptions, null, new
       IMqttActionListener() {
                @Override
                public void onSuccess(IMqttToken asyncActionToken) {
                    DisconnectedBufferOptions disconnectedBufferOptions
       = new DisconnectedBufferOptions();
                    disconnectedBufferOptions.setBufferEnabled(true);
                    disconnectedBufferOptions.setBufferSize(100);
                    disconnectedBufferOptions.setPersistBuffer(false);

       disconnectedBufferOptions.setDeleteOldestMessages(false);

       mqttAndroidClient.setBufferOpts(disconnectedBufferOptions);
                    subscribeToTopic("MensajesAplicacion");
                    subscribeToTopic("DispositivosCita");
                    subscribeToTopic("EntrarPaciente");
                    subscribeToTopic("Aparcamientos");
                }
                @Override
                public void onFailure(IMqttToken asyncActionToken,
       Throwable exception) {
                        Log.w("Mqtt", "Fallo en la conexión a: " +
       serverUri + exception.toString());
                    Log.w("Mqtt", "=========");
                }
            });
        } catch (MqttException ex){
            ex.printStackTrace();
        }
    }
```

```
    private void subscribeToTopic(String topic) {
        try {
            mqttAndroidClient.subscribe(topic, 0, null, new
        IMqttActionListener() {
                @Override
                public void onSuccess(IMqttToken asyncActionToken) {
                    Log.w("Mqtt","Suscrito!");
                }
                @Override
                public void onFailure(IMqttToken asyncActionToken,
        Throwable exception) {
                    Log.w("Mqtt", "Fallo en la suscripcion!");
                }
            });
        } catch (MqttException ex) {
            System.err.println("Excepcion en la suscripcion");
            ex.printStackTrace();
        }
    }
}
```

Para manejar los mensajes que devuelve este método se implementa el método **`startMqtt()`**, que maneja el *callback* del servicio:

```
  private void startMqtt() {
       Toast.makeText(MainActivity.this, "Conectando con el servidor
MQTT...", Toast.LENGTH_SHORT).show();
    mqttHelper = new MqttHelper(getApplicationContext());
    mqttHelper.setCallback(new MqttCallbackExtended() {
        @Override
        public void messageArrived(String topic, MqttMessage
        mqttMessage) throws Exception {
            int posicion, posicion1;
            String linea1, linea2, idrecibido, idandroid;
            Log.w("Debug", mqttMessage.toString());
            if (topic.equals("MensajesAplicacion")) {
                posicion=mqttMessage.toString().indexOf("=");
                posicion1=mqttMessage.toString().indexOf("%");
                if (posicion>0) {
             idrecibido=mqttMessage.toString().substring(0,posicion);
                    idandroid =
        android.provider.Settings.System.getString(getContentResolver(),
        android.provider.Settings.System.ANDROID_ID);
                    if (idrecibido.equals(idandroid)) { // Si el
                    destinatario es este dispositivo

linea1=mqttMessage.toString().substring(posicion+1, posicion1);

linea2=mqttMessage.toString().substring(posicion1+1,mqttMessage.toStrin
g().length());
                        mensajes.setText(linea1);
                        mensajes1.setText(linea2);
```

```
                    }
                }
            }
            else if (topic.equals("DispositivosCita")) {
                posicion=mqttMessage.toString().indexOf("=");
                posicion1=mqttMessage.toString().indexOf("%");
                if (posicion>0) {

      idrecibido=mqttMessage.toString().substring(0,posicion);
                    idandroid =
      android.provider.Settings.System.getString(getContentResolver(),
      android.provider.Settings.System.ANDROID_ID);
                    if (idrecibido.equals(idandroid)) { // Si el
      destinatario es este dispositivo
      linea1=mqttMessage.toString().substring(posicion+1,   posicion1);

linea2=mqttMessage.toString().substring(posicion1+1,mqttMessage.toStrin
g().length());
                                        mensajes.setText("Tiene cita en:
".concat(linea1));
                        mensajes1.setText("A las ".concat(linea2));
                        actualizarEstado(estado,1);
                    }
                }
            }
            else if (topic.equals("EntrarPaciente")) {
                    posicion=mqttMessage.toString().indexOf("=");
                    if (posicion>0) {

      idrecibido=mqttMessage.toString().substring(0,posicion);
                                                        idandroid  =
      android.provider.Settings.System.getString(getContentResolver(),
      android.provider.Settings.System.ANDROID_ID);
                            if (idrecibido.equals(idandroid)) { // Si el
destinatario es este dispositivo

      linea1=mqttMessage.toString().substring(posicion+1,
      mqttMessage.toString().length());
                                mensajes.setText("Por favor, pase a la
                           consulta:");
                            mensajes1.setText(linea1);
                            actualizarEstado(estado,4);
                        }
                    }
            }
            else if (topic.equals("Aparcamientos")) {
                aparcamientos.setText(mqttMessage.toString());
            }
        }
    });
}
```

Para publicar mensajes se utiliza el método **publish(String topic, String message)**:

```
public void publish(String topic, String message) {
    MqttMessage mqttMessage = new MqttMessage();
    mqttMessage.setQos(0);
    mqttMessage.setRetained(false);
    mqttMessage.setPayload(message.getBytes());
    try {
        mqttHelper.mqttAndroidClient.publish(topic, mqttMessage);
    }
    catch (Exception e) {
        System.out.println("Error " + e.getMessage());
    }
  }
```

Para buscar los beacons se ha implementado el servicio **BackgroundScanService:**

```
package service;
import android.app.Service;
...
public class BackgroundScanService extends Service {
    public static final String TAG = "BackgroundScanService";
          public static final String ACTION_DEVICE_DISCOVERED =
"DeviceDiscoveredAction";
    public static final String EXTRA_DEVICE = "DeviceExtra";
            public static final String EXTRA_DEVICES_COUNT =
"DevicesCountExtra";
    public static final String EXTRA_NAME = "DeviceId";
    private static final long TIMEOUT = TimeUnit.SECONDS.toMillis(30);
    private final Handler handler = new Handler();
    private ProximityManager proximityManager;
     private boolean isRunning; // Flag que indica si el servicio ya
está activo
    private int devicesCount; // Dispositivos encontrados
    private String nombre; // Nombre del dispositivo encontrado
    @Override
    public void onCreate() {
        super.onCreate();
        setupProximityManager();
        isRunning = false;
    }
    private void setupProximityManager() {
        //Crear la instancia del proximity manager
        proximityManager = ProximityManagerFactory.create(this);
        //Configurar las opciones del proximity manager
        proximityManager.configuration()
                    //Se usa ranging para escaneo contínup o MONITORING
para escanear a intervalos
                .scanPeriod(ScanPeriod.RANGING)
                      //Se usa BALANCED para mejorar la relación entre
rendimiento y consumo
                .scanMode(ScanMode.BALANCED);
```

```
        //Poniendo a la escucha los listeners para ebeacons
        proximityManager.setIBeaconListener(createIBeaconListener());
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        //Mira si el servicio ya está activo
        if (isRunning) {
             Toast.makeText(this, "El servicio ya se está ejecutando.",
Toast.LENGTH_SHORT).show();
            return START_STICKY;
        }
        startScanning();
        isRunning = true;
        return START_STICKY;
    }
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
    private void startScanning() {
        proximityManager.connect(new OnServiceReadyListener() {
            @Override
            public void onServiceReady() {
                proximityManager.startScanning();
                devicesCount = 0;
                 Toast.makeText(BackgroundScanService.this, "Búsqueda de
beacons iniciada.", Toast.LENGTH_SHORT).show();
            }
        });
        stopAfterDelay();
    }
    private void stopAfterDelay() {
        handler.postDelayed(new Runnable() {
            @Override
            public void run() {
                proximityManager.disconnect();
                stopSelf();
            }
        }, TIMEOUT);
    }
    private IBeaconListener createIBeaconListener() {
        return new SimpleIBeaconListener() {
            @Override
                 public void onIBeaconDiscovered(IBeaconDevice ibeacon,
IBeaconRegion region) {
                onDeviceDiscovered(ibeacon);
                                Log.i(TAG,  "onIBeaconDiscovered:  "  +
ibeacon.toString());
            }
        };
    }
    private void onDeviceDiscovered(RemoteBluetoothDevice device) {
        devicesCount++;
```

```
        //Envía un broadcast con el dispositivo descubierto
        Intent intent = new Intent();
        intent.setAction(ACTION_DEVICE_DISCOVERED);
        intent.putExtra(EXTRA_DEVICE, device);
        nombre=device.getUniqueId();
            Toast.makeText(this, "'Encontrado beacon: ".concat(nombre),
Toast.LENGTH_SHORT).show();
        intent.putExtra(EXTRA_NAME, nombre);
        intent.putExtra(EXTRA_DEVICES_COUNT, devicesCount);
        sendBroadcast(intent);
    }
    @Override
    public void onDestroy() {
        handler.removeCallbacksAndMessages(null);
        if (proximityManager != null) {
            proximityManager.disconnect();
            proximityManager = null;
        }
         Toast.makeText(BackgroundScanService.this, "Búsqueda de beacons
detenida.", Toast.LENGTH_SHORT).show();
        super.onDestroy();
    }
}
```

Este método devuelve los dispositivos descubiertos en un *broadcast*, cuya recepción se implementa en la *activity* principal mediante el método **scanningBroadcastReceiver**:

```
private final BroadcastReceiver scanningBroadcastReceiver = new
BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
                // Recepción de los dispositivos descubiertos por el
servicio
                                                    int devicesCount =
intent.getIntExtra(BackgroundScanService.EXTRA_DEVICES_COUNT, 0);
                                        RemoteBluetoothDevice device =
intent.getParcelableExtra(BackgroundScanService.EXTRA_DEVICE);
                                                                String
nombre=intent.getStringExtra(BackgroundScanService.EXTRA_NAME);
            Log.w("Beacon", String.format("Encontrado: %s", nombre));
            if (estado==2) {
                if (devicesCount>0) { // Se ha encontrado algún beacon
                        if (nombre.equals(nombrebeacon)){ // Si es mi
beacon
                       //estado=3; // Está en el área del ebeacon
                        actualizarEstado(estado,3); // Está en el área
del beacon
                    }
                }
                else if (estado==3) {
                      if (devicesCount==0) { // Se ha encontrado algún
beacon
                        actualizarEstado(estado,2); // Está en el área
del beacon
```

```
                        //estado=2; // Salgo del área del ebeacon
                        }
                    }
                }
                 //Toast.makeText(MainActivity.this, "Encontrado: "+nombre,
Toast.LENGTH_SHORT).show();
            }
        };
}
```

4.3.3. Implementación del *middleware*

El módulo intermedio estará implementado en Delphi al igual que la aplicación de gestión de la clínica.

Se creará una interfaz simple y útil, con controles para conectarse/desconectarse al servidor MQTT y para suscribirse/anular suscripciones, y con un cuadro de texto para mostrar la actividad de la aplicación:

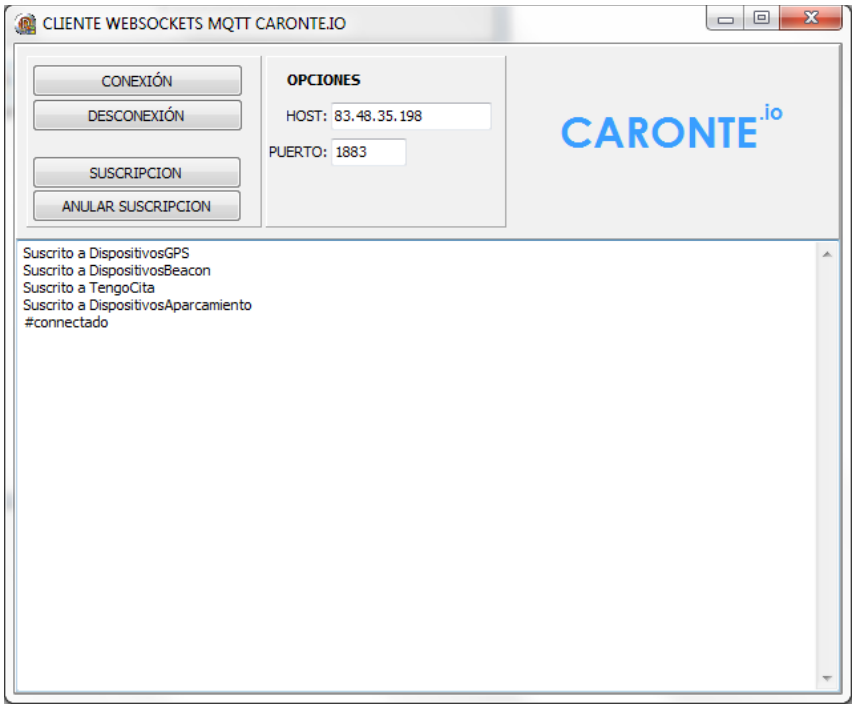


Fig. 35: Interfaz del cliente websockets MQTT

Para comunicarse con el servidor MQTT se utiliza la versión de prueba de los componentes sgcWebSockets, disponibles en (Esegece, 2017).

De entre todos ellos se usan solo dos: el componente `TsgcWebSocketClient` y el componente `TsgcWSPClient_MQTT`, que hay que enlazar mediante la propiedad *Client* de `TsgcWSPClient_MQTT` para después pasar a configurar con los parámetros de comunicaciones del servidor MQTT.

Las principales funciones y procedimientos de esta aplicación son las siguientes:

- Procedimientos **`BtConectarClick`** y **`BtdeSCONECTARClick`** activados por el evento OnClick de los botones CONECTAR y DESONECTAR respectivamente. Se encargan de la conexión y desconexión de la aplicación al servidor MQTT:

```
procedure TfrmClientPROTOCOL.BtConectarClick(Sender: TObject);
begin
WSClient.Port := StrToInt(txtDefaultPort.text);
WSClient.Host := txtHost.text;
WSClient.TLS := fALSE;
WSClient.Proxy.Enabled := False;
WSClient.Extensions.DeflateFrame.Enabled := False;
WSClient.Active := True;
if WSClient.Active then
      pnlClientOptions.Enabled := False;
end;

procedure TfrmClientPROTOCOL.BTdeSCONECTARClick(Sender: TObject);
begin
WSClient.Active := False;
if not WSClient.Active then
      pnlClientOptions.Enabled := True;
end;
```

- Procedimientos **`BtSuscripcionClick`** y **`BtAnularSuscripcionClick`**, activados por el evento OnClick de los botones SUSCRIPCIÓN y ANULAR SUSCRIPCIÓN respectivamente. Suscribe o anulan las suscripciones a las colas DispositivosGPS, DispositivosSEspera, TengoCita y DispositivosAparcamiento:

```
procedure TfrmClientPROTOCOL.BtSuscripcionClick(Sender: TObject);
begin
MQTT.Subscribe('DispositivosGPS');
memolog.Lines.Add('Suscrito a DispositivosGPS');
MQTT.Subscribe('DispositivosSEspera');
memolog.Lines.Add('Suscrito a DispositivosSEspera');
MQTT.Subscribe('TengoCita');
memolog.Lines.Add('Suscrito a TengoCita');
MQTT.Subscribe('DispositivosAparcamiento');
memolog.Lines.Add('Suscrito a DispositivosAparcamiento');
end;

procedure TfrmClientPROTOCOL.BTAnularSuscripcionClick(Sender: TObject);
begin
MQTT.UNSubscribe('DispositivosGPS');
memolog.Lines.Add('Anulada suscripción a DispositivosGPS');
MQTT.UNSubscribe('DispositivosSEspera');
memolog.Lines.Add('Anulada suscripción a DispositivosSEspera');
MQTT.UNSubscribe('TengoCita');
memolog.Lines.Add('Anulada suscripción a TengoCita');
end;
```

- Procedimiento **`MQTTMQTTPublish`**. Lleva a cabo las acciones necesarias según los mensajes recibidos por las distintas colas. SCOB PU 2017 utiliza el campo ETAPA para distinguir entre visitas en el turno de mañanas o de tardes, así que si la consulta desde la app se hace por la mañana, la aplicación buscará solo las citas entre las de antes del mediodía, y procederá de forma análoga cuando la aplicación se conecte por la tarde:

```
procedure TfrmClientPROTOCOL.MQTTMQTTPublish(Connection:
TsgcWSConnection; aTopic, aText: string);
var
	s: string;
	q, q1: Tadoquery;
	Comando: string;
	id: string;
	etapa: string;
	h,m,se,ms: word;
	mensaje: string;
begin
q:=tadoquery.Create(nil);
q.Connection:=AdoCeMysql;
q1:=tadoquery.Create(nil);
q1.Connection:=AdoCeMysql;
memolog.Lines.Add('Recibido por: #'+aTopic+'# el mensaje: '+atext);
if atopic='DispositivosGPS' then
	begin
	// Los mensajes deben ser del tipo 1=Id si entra, 0=Id si sale
	comando:=copy(aText,1,1);
	Id:=copy(Atext,3,(length(AText)-2));
	s:='select apellidos, nombre, fecha_N from pacientes where
	IdDispositivo="'+id+'"';
	q.SQL.Clear;
	q.SQL.Add(s);
	q.Open;
	if q['apellidos']<>null then
		// Busco la visita
		begin
		decodetime(now,h,m,se,ms);
		if h>15 then
			etapa:='T'
		else
			etapa:='M';
		if comando='1' then // Entra
			begin
			// Actualizo el estado de la visita
			s:='update visitas set Vino="G" where Vino="N" and
			apellidos="'+q['apellidos']+'" and nombre="'
			// Le indico dónde está la sala de espera
			mensaje:=id+'='+'Diríjase a:
			'+SalaDeEspera+'%Retraso: '+retraso1;
			memolog.Lines.Add('Publicando en
			#MensajesAplicacion#: '+mensaje);
			MQTT.Publish('MensajesAplicacion', mensaje,
			mtqsAtMostOnce, False);
```

```
                    end
              else if comando='0' then
                    s:='update visitas set Vino="N" where Vino="G" and
                    apellidos="'+q['apellidos']+'" and nombre="'
              end;
       q1.SQL.Clear;
       q1.SQL.Add(s);
       memolog.Lines.Add('Ejecutando: '+s);
       q1.ExecSQL;
       end
else if atopic='DispositivosSEspera' then
       begin
       // Los mensajes deben ser del tipo 1=Id si entra, 0=Id si sale
       comando:=copy(aText,1,1);
       Id:=copy(Atext,3,(length(AText)-2));
       s:='select apellidos, nombre, fecha_N from pacientes where
       IdDispositivo="'+id+'"';
       q.SQL.Clear;
       q.SQL.Add(s);
       q.Open;
       if q['apellidos']<>null then
              // Paciente localizado
              // Busco la visita
              begin
              decodetime(now,h,m,se,ms);
              if h>15 then
                    etapa:='T'
              else
                    etapa:='M';
              if comando='1' then // Entra
                    begin
                    s:='update visitas set Vino="S" where Vino="G" and
                    apellidos="'+q['apellidos']+'" and nombre="'
                    mensaje:=id+'=Espere aquí, por favor'+'%Retraso:
                    '+retraso2;
                    memolog.Lines.Add('Publicando en
                    #MensajesAplicacion#: '+mensaje);
                    MQTT.Publish('MensajesAplicacion', mensaje,
                    mtqsAtMostOnce, False);
                    end
              else if comando='0' then
                    s:='update visitas set Vino="G" where Vino="S" and
                    apellidos="'+q['apellidos']+'" and nombre="'
              end;
       q1.SQL.Clear;
       q1.SQL.Add(s);
       memolog.Lines.Add('Ejecutando: '+s);
       q1.ExecSQL;
       end
else if atopic='TengoCita' then
       // El mensaje contiene solo el id
       begin
       id:=aText;
```

```
      s:='select apellidos, nombre, fecha_N from pacientes where
      IdDispositivo="'+id+'"';
      q.SQL.Clear;
      q.SQL.Add(s);
      q.Open;
      if q['apellidos']<>null then
            begin
            decodetime(now,h,m,se,ms);
            if h>15 then
                  etapa:='T'
            else
                  etapa:='M';
            s:='select hora from visitas where (vino="N" or vino="G")
            and etapa="'+etapa+'" and apellidos="'
            q1.SQL.Clear;
            q1.SQL.Add(s);
            q1.Open;
            if q1['hora']<>null then
                  begin
                  // Si tiene cita, le respondo con un mensaje en
                  // DispositivosCita del tipo id=consulta%hora
                  decodetime(q1['hora'],h,m,se,ms);
                  mensaje:= id+'=Clinica Fort%'+inttostr(h)
                  +':'+inttostr(m);
                  memolog.Lines.Add('Publicando en
                  #DispositivosCita#: '+mensaje);
                  MQTT.Publish('DispositivosCita',mensaje,
                  mtqsAtMostOnce, False);
                  end;
            end;
      end;
if atopic='DispositivosAparcamiento' then
      begin
      memolog.Lines.Add('Publicando      en      #Aparcamientos#:
      '+ObtenerAparcamientos);
      MQTT.Publish('Aparcamientos', ObtenerAparcamientos,
      mtqsAtMostOnce, False);
      end;
q.free;
q1.Free;
end;
```

- Función **`ObtenerAparcamientos`**. Llamada por **`MQTTMQTTPublish`** al recibir un mensaje en la cola DispositivosAparcamiento, obtiene los aparcamientos regulados cercanos a las coordenadas definidas en las constantes `x_hospital` e `y_hospital`:

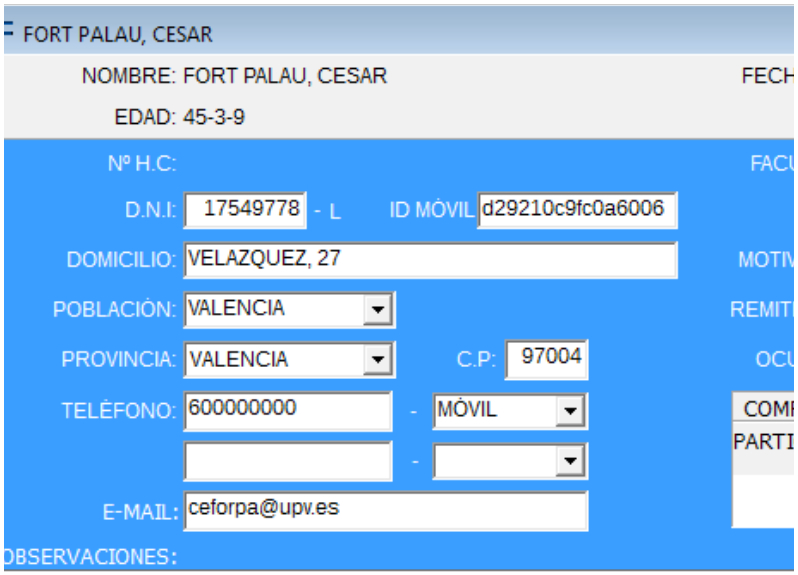
```
function TfrmClientPROTOCOL.ObtenerAparcamientos: ansistring;
var
      s,s1: string;
      q, q1: Tadoquery;
      Comando: string;
```

```
      id: string;
      etapa: string;
      h,m,se,ms: word;
      mensaje: string;
      begin
q:=tadoquery.Create(nil);
q.Connection:=AdoCeMysql;
q.SQL.Clear;
s:='SELECT      nombre,direccion      FROM      aparcamientos      WHERE
SQRT(((X-'+x_hospital+')*(X-'+x_hospital+'))+((q.SQL.Add(s);
q.Open;
s1:='';
q.First;
while not q.Eof do
      begin
      s1:=s1+q['nombre']+'-'+q['direccion']+'. ';
      q.Next
      end;
q.Free;
result:=s1;
end;
```

4.3.4. Implementación de los cambios en SCOB PU 2017

Tal y como se indicaba en el apartado 4.2.4. Diseño de las modificaciones en la aplicación de la clínica, las modificaciones que es necesario hacer en dicha aplicación son las siguientes:

- Añadir a los datos de filiación del paciente el campo ID MOVIL:



- Instalar el componentes TsgcWebSocketClient y TsgcWSPClient_MQTT mencionados en el apartado 4.3.3. Implementación del middleware.

- Modificar el código Delphi que se encarga de que se muestre en el grid de visitas los estados para contemplar el nuevo estado G, y que al detectar que una visita tiene ese estado muestre el icono correspondiente  . El procedimiento es el que se encarga de actualizar las celdas del *grid* de las visitas, **`DBGVisitasDrawColumnCell`**:

```
procedure TPonerCita.DBGVisitasDrawColumnCell(Sender: TObject; const
Rect: TRect; DataCol: Integer;
...
case column.Index of
6: begin
      with dbgvisitas.canvas do
      begin
      font.color:=clred;
      if dbgvisitas.datasource.dataset['Vino']<>null then
            begin
            if dbgvisitas.datasource.dataset['Vino']='S' then
            begin
            font.Name:='Windings';
            font.Color:=clred;
            TextOut(Rect.Left+trunc(dbgvisitas.columns
            [column.index].width/2)-trunc(font.size
            end
      else if dbgvisitas.datasource.dataset['Vino']='D' then
            begin
            font.Name:='Windings';
            font.Color:=clgreen;
            TextOut(Rect.Left+trunc(dbgvisitas.columns
            [column.index].width/2)-trunc(font
            end
      else if dbgvisitas.datasource.dataset['Vino']='G' then
            // Está en el área GPS
            begin
            font.Color:=clblack;
            font.Name:='Building';
            TextOut(Rect.Left+trunc(dbgvisitas.columns
            [column.index].width/2)-trunc(font
            end
...
```

- Modificar el código Delphi del procedimiento encargado de que se ejecuten la acción de llamar a un paciente. Se añadirá la publicación por MQTT del mensaje: [Id del dispositivo]=[Lugar de la consulta].

Esta acción se lleva a cabo haciendo click con el botón derecho del ratón sobre las visitas, de modo que aparece el menú emergente:

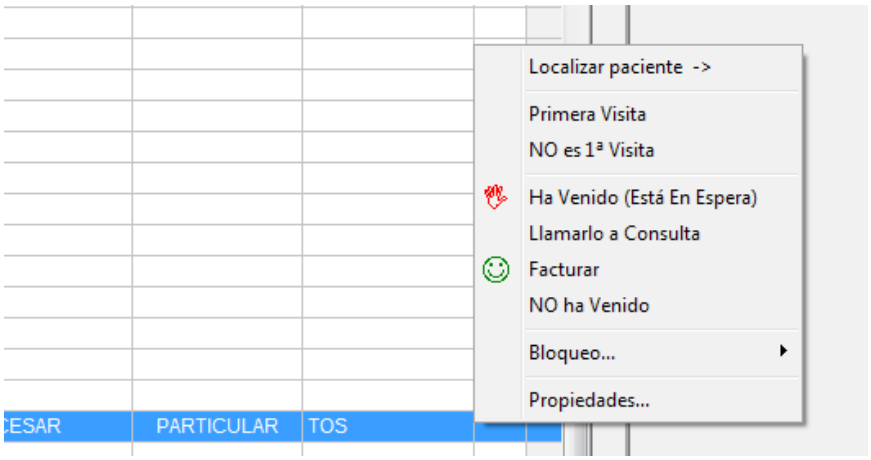


Fig. 36: Menú emergente de las visitas de SCOB PU 2017

El procedimiento a modificar es el que se ejecuta al hacer click sobre Llamarlo a Consulta, tratándose en este caso de **`Llamarloaconsulta1Click`**. Se busca en la base de datos el Id del dispositivo y se publica en la cola LlamarPaciente el mensaje con su Id y la sala de consulta, definida en la variable SalaDeConsulta:

```
procedure TPonerCita.Llamarloaconsulta1Click(Sender: TObject);
...
// Añadirlo a la cola MQTT
// Busco el Id del paciente
t:='Select IdDispositivo from pacientes where apellidos="' +
visitasdeldia['Apellidos'] + '" and nombre="';
q.sql.Clear;
q.SQL.Add(t);
q.Open;
// Si tiene dispositivo dado de alta
if q['IdDispositivo']<>null then
      begin
      m:=q['IdDispositivo']+'='+SalaDeConsulta;
      MQTT.Publish('EntrarPaciente', m,mtqsAtMostOnce, False);
      end;
q.Free;
...
end;
```

4.4. Construcción y pruebas

Para el desarrollo de este proyecto, dada la extensión y complejidad del mismo, y dado también que el equipo de trabajo se reducía a una única persona, se ha seguido una estrategia incremental en la que ha tenido una importancia clave la utilización de un cliente MQTT para hacer pruebas parciales.

El cliente seleccionado ha sido MQTTlens para Google Chrome.

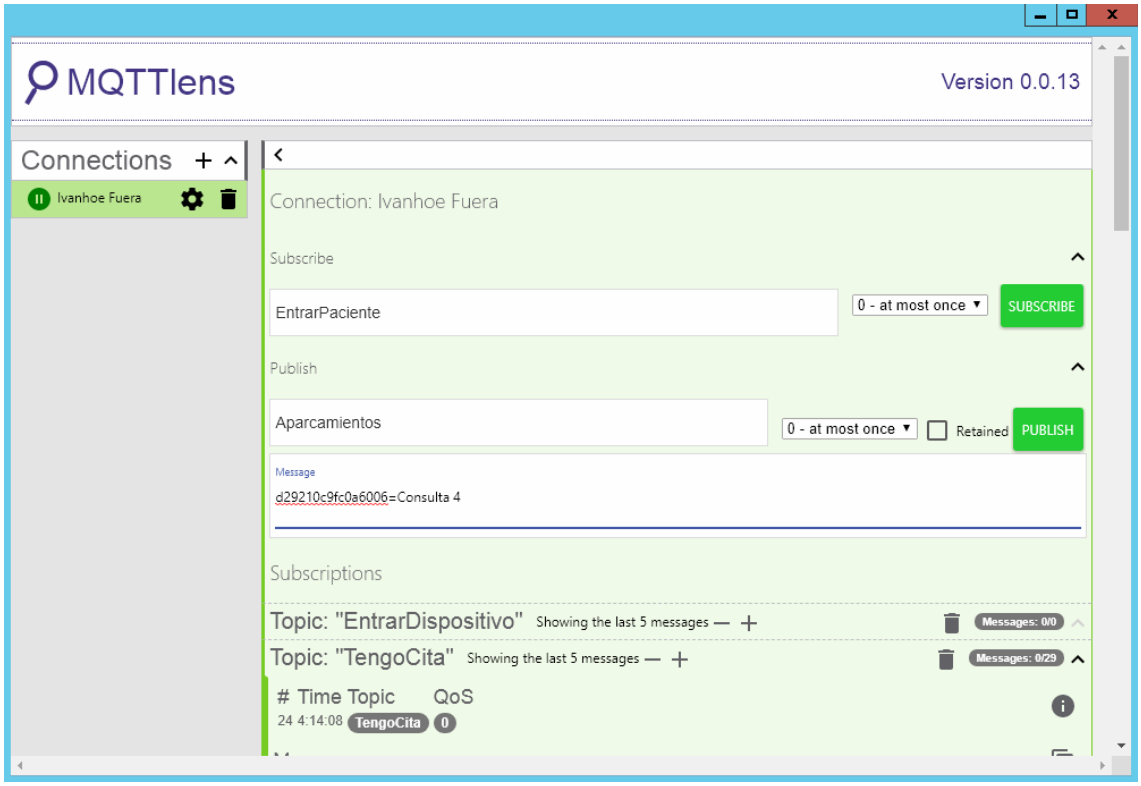


Fig. 37: MQTTlens

Así, se hicieron primero las modificaciones en la aplicación de la clínica, se creó una primera versión del middleware, y se probaron ambas con MQTTlens ejecutándose en otra máquina fuera del centro médico.

Del mismo modo, se implementó la app haciendo pruebas de forma que con MQTTlens se simulaban las respuestas de la clínica, y en el siguiente orden:

1. Implementación de la funcionalidad básica de publicación y suscripción de mensajes al servidor MQTT
2. Implementación de la geolocalización
3. Implementación de la búsqueda de beacons
4. Implementación de la funcionalidad de solicitar y mostrar los aparcamientos cercanos

Finalmente se hicieron las pruebas de integración y se hicieron los ajustes necesarios para eliminar los errores encontrados.

4.5. Aspectos sobre la seguridad de la información

En el presente proyecto se ha tenido en cuenta los requisitos recogidos en las siguientes imposiciones legales:

1. Ley Orgánica 15/1999, de 13 de diciembre de Protección de Datos de Carácter Personal (BOE-A-1999-23750, 1999).
2. Real Decreto 1720/2007, por el que se aprueba el Reglamento de Desarrollo de la Ley 15/1999 de Protección de Datos de Carácter Personal (BOE-A-2008-979, 2007).
3. Resoluciones vigentes en materia de seguridad de datos (AGPD-Resoluciones, 2017).
4. Recomendaciones vigentes de la Agencia Española de Protección de Datos (AGPD-Recomendaciones, 2017).
5. Ley orgánica 34/2002 de 11 de Julio, Servicios de la Sociedad de la Información y de Comercio Electrónico (BOE-A-2002-13758, 2002).
6. Real decreto 1332/94 de 20 de junio por el que se desarrollan algunos preceptos de la Ley Orgánica (BOE-A-1994-14121, 2004).

Es por esto que la información que se transmite a través de las colas MQTT en ningún caso contiene datos personales, ya que la forma de identificar a los pacientes es mediante el Id de su dispositivo móvil. La identificación del Id del dispositivo con el resto de datos del paciente se hace en la propia aplicación de gestión del centro médico, así que no se vulnera ninguna de las disposiciones legales.

En cualquier caso, también es posible implementar MQTT utilizando TLS (Transport Layer Security) y SSL (Secure Sockets Layer) para reforzar la seguridad del sistema.

5. Conclusiones. Líneas futuras

Dijimos en 1.2. Objetivos del proyecto que el principal objetivo era mejorar el proceso de llegada de un paciente a un centro médico, y además hacerlo mediante el diseño e implementación de un sistema de información heterogéneo.

En la propuesta ofrecida por el presente proyecto, que efectivamente incorpora un sistema de información formado por dispositivos de distinta índole: servidores, estaciones de trabajo, smartphones y beacons, se elimina los siguientes elementos que suponen un coste en tiempo:

1. Que el paciente deba buscar aparcamiento por sí mismo, ya que se le indica dónde tiene a su disposición aparcamientos regulados.
2. Que deba preguntar en la recepción del centro médico dónde se encuentra la consulta en la que está citado, ya que se le indica dónde debe esperar a ser atendido.
3. Tener que anunciar su llegada al personal de recepción, ya que su llegada es anunciada automáticamente al llegar a la sala de espera.
4. Que tenga alguna duda sobre dónde tiene personarse que ir al ser llamado a la consulta, ya que se le indica expresamente mediante un mensaje.
5. Que el centro sanitario pueda no tener conocimiento de que el paciente está en espera, bien por no haber anunciado su llegada, o por error del propio centro.

Además, volviendo a la relación de inconvenientes del sistema de citación tradicional planteados en 2. Descripción del problema, podemos revisar cuales de ellos se palían o eliminan completamente con esta nueva forma de proceder, y cuales no se ven afectados por la propuesta, marcándolos con ✓ o ✗ respectivamente:

1. El paciente puede haber olvidado o extraviado la información de la cita: dónde y a qué hora debe presentarse. ✓
2. La cita puede requerir la aportación por parte del paciente de documentación que puede haber olvidado. ✗
3. El paciente se puede encontrar con diferentes circunstancias que pueden retrasarle: congestión de tráfico, búsqueda de aparcamiento, huelgas de transporte, etc. ✓
4. El médico puede llevar retraso. Es habitual que en las clínicas se cite a los pacientes con una frecuencia algo inferior al necesario con el objetivo de maximi-

zar la productividad, con lo que los retrasos se van acumulando de forma progresiva a lo largo de la jornada. ✗

5. Hay médicos que también atienden urgencias o incluso partos mientras pasan consulta, con lo que se producen nuevos retrasos. ✗
6. El paciente no es conocedor del retraso que lleva el médico hasta que no llega a la clínica, con lo que después del esfuerzo realizado por llegar a su hora se ve obligado a esperar. ✓
7. No se suele facilitar al paciente información veraz y objetiva del tiempo que debe esperar a ser atendido. ✓
8. Si en un momento dado un paciente no acude a su cita, la clínica no tiene ninguna información sobre la ubicación del paciente. ✓
9. La clínica no tiene una forma eficaz de comprobar si el paciente sabe que tiene cita. ✓
10. Cuando el paciente llega a una clínica, sobretodo por vez primera, no sabe a quién debe dirigirse para comunicar su llegada, con lo que se producirán nuevas esperas. ✓
11. Un paciente no tiene una forma sencilla y eficaz de comunicar a la clínica la imposibilidad de acudir a una cita. ✗
12. Cuando un paciente está en una sala de espera junto con otros pacientes, llega un momento en el que debe ser llamado a consulta, y para preservar la privacidad se debe llevar a cabo esta llamada de forma de forma discreta sin revelar ninguno de sus datos personales. Este aspecto es especialmente delicado en especialidades como cirugía plástica, urología, ginecología, o psiquiatría. ✓
13. El hecho de que pacientes con diversas patologías compartan una sala de espera durante un tiempo prolongado puede causarles problemas de estrés que se añaden a las dolencias propias de su problema médico. ✓
14. Los retrasos producen en los pacientes insatisfacción y frustración, con lo que la sensación de calidad percibida es baja, independientemente de la efectividad del tratamiento médico. ✓
15. El prolongar más de lo imprescindible la estancia de un paciente en un hospital aumenta de forma innecesaria el riesgo de que contraiga algún tipo de infección intrahospitalaria. ✓

Todo ello suma un total de 11 de 13 inconvenientes a los que se les ha encontrado mejora, o sea a un **85%** de ellas, con lo que junto con el previsible ahorro de tiempo comen-

tado anteriormente la presente propuesta promete ser eficiente y cumple con los objetivos marcados.

No obstante, el sistema puede ser mejorado sustancialmente con las siguientes ampliaciones:

1. Se debería mejorar la seguridad utilizando validación de credenciales y encriptando las comunicaciones mediante SSL/TLS.
2. Se podría mejorar la seguridad incorporando a la app un sistema de reconocimiento biométrico.
3. Se podría añadir la funcionalidad de obtener o anular una cita a través de la App.
4. Se podría llevar a cabo una integración del sistema con plataformas de pago y también con las plataformas online de las compañías aseguradoras para que la contraprestación de los servicios médicos fuera también automática.
5. Actualmente los datos obtenidos de las smart cities deben importarse desde los ficheros CSV o KML a la base de datos de la clínica, con lo que hay que repetir esta importación de forma regular para actualizar los datos. Ello no sería necesario si se implementara el acceso online al repositorio de datos de las smart cities en tiempo real.
6. Los datos de acceso al repositorio de la smart city deberían ser parametrizables, de forma que el sistema fuera sencillo de implementar en cualquier centro médico que estuviera ubicado en una población con dichas facilidades.
7. Se podrían incorporar nuevas funcionalidades incorporando más datos disponibles en las smart cities, como:
    - El estado del tráfico en tiempo real
    - La búsqueda de aparcamientos no regulados, para personas con movilidad reducida, o para motos en las inmediaciones del centro sanitario
    - La búsqueda de puntos de recarga de vehículos eléctricos en los alrededores del centro sanitario
    - Las paradas cercanas de autobús o metro del servicio municipal de transporte
    - Las estaciones del servicio municipal de alquiler de bicicletas cercanas al centro sanitario
8. Se podría añadir más beacons en puntos estratégicos del hospital, para guiar al paciente en su recorrido desde la puerta hasta la sala de espera.

9. Se podría añadir al sistema una encuesta de satisfacción que el paciente cumplimentaría al abandonar el centro médico.
10. Se podrían poner servidores MQTT redundantes para aumentar la tolerancia a fallos.
11. Se podría incorporar una API a la que se pudieran conectar los sistemas de gestión hospitalarios.

Además, aunque el presente proyecto se ha centrado en plantear una solución para un centro médico, el sistema propuesto es perfectamente susceptible de ser implantado en cualquier entorno que conlleve citación de personas: centros públicos de atención al usuario, tutorías académicas, despachos de abogados, etc.

6. Bibliografía y referencias

DuBois, 2009: DuBois, P.(2009). La biblia de MySQL. Rumford, ME (EE. UU.): Anaya Multimedia

Wikipedia-MySQL, 2017: The Wikimedia Foundation, Inc., MySQL. <https://es.wikipedia.org/wiki/MySQL> [Consulta: 2 de Julio de 2017]

Wikipedia-Oracle, 2017: The Wikimedia Foundation, Inc., Oracle Database. <https://es.wikipedia.org/wiki/Oracle_Database> [Consulta: 4 de Julio de 2017]

Oracle, 2010: Oracle Corp., Familia de productos Oracle Database 11g. <http://www.oracle.com/technetwork/es/database/enterprise-edition/documentation/productos-oracle-database-11g-2247590-esa.pdf> [Consulta: 2 de Julio de 2017]

Wikipedia-SQLServer, 2017: The Wikimedia Foundation, Inc., Microsoft SQL Server. <https://es.wikipedia.org/wiki/Microsoft_SQL_Server> [Consulta: 2 de Julio de 2017]

Kantar, 2017: ComTech, Smartphone OS sales market share. <https://www.kantarworldpanel.com/global/smartphone-os-market-share/> [Consulta: 4 de Julio de 2017]

Wikipedia-Android, 2017: The Wikimedia Foundation, Inc., Android. <https://es.wikipedia.org/wiki/Android> [Consulta: 3 de Julio de 2017]

Android.com, 2017: Android.com, Android Studio. The Official IDE for Android. <https://developer.android.com/studio/index.html> [Consulta: 3 de Julio de 2017]

Wikipedia-iOS, 2017: The Wikimedia Foundation, Inc., iOS. <https://es.wikipedia.org/wiki/IOS> [Consulta: 3 de Julio de 2017]

Apple.com, 2017: Apple Inc., XCode - Features. <https://developer.apple.com/xcode/features/> [Consulta: 3 de Julio de 2017]

Wikipedia-Servicios Web, 2017: Fundación Wikimedia, Servicio web - Wikipedia, la enciclopedia libre. <https://es.wikipedia.org/wiki/Servicio_web> [Consulta: 4 de Julio de 2017]

Wikipedia-REST, 2017: Fundación Wikimedia, Inc., Transferencia de Estado Representacional - Wikipedia, la enciclopedia libre. <> [Consulta: 4 de Julio de 2017]

Marques, 2013: Marqués, A., Conceptos sobre APIs REST. <http://asiermarques.com/2013/conceptos-sobre-apis-rest/> [Consulta: 4 de Julio de 2017]

IBM, 2010: International Business Machines Corporation (IBM), Eurotech, MQTT V3.1 Protocol Specification. <http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html> [Consulta: 30 de Junio de 2017]

HiveMQ, 2017: HiveMQ- Enterprise MQTT Broker, MQTT Essentials Part 9: Last Will and Testament. <http://www.hivemq.com/blog/mqtt-essentials-part-9-last-will-and-testament> [Consulta: 30 de Junio de 2017]

HiveMQ, 2017-2: dc-square GmbH, MQTT Essentials Special: MQTT over WebSockets. <http://www.hivemq.com/blog/mqtt-essentials-special-mqtt-over-websockets> [Consulta: 31 de Agosto de 2017]

HiveMQ, 2017-3: dc-square GmbH, HiveMQ. <http://www.hivemq.com/hivemq/> [Consulta: 31 de Agosto de 2017]

Mosquitto, 2017: mosquitto.org, An Open Source MQTT v3.1 Broker. <> [Consulta: 31 de Agosto de 2017]

Eclipse, 2017: The Eclipse Foundation, Mosquitto. <http://www.eclipse.org/proposals/technology.mosquitto/> [Consulta: 31 de Agosto de 2017]

Amazon, 2017: Amazon Web Services, Inc., AWS IoT - Amazon Web Services. <https://aws.amazon.com/es/iot> [Consulta: 31 de Agosto de 2017]

ECMA, 2013: ECMA International(2013). The JSON Data Interchange Format. Ginebra (Suiza): ECMA International

Ben-Kiki, 2009: Ben-Kiki, O, Evans, C. y döt Net, I., YAML Ain't Markup Language (YAML™) Version 1.2. <http://www.yaml.org/spec/1.2/spec.html> [Consulta: 30 de Junio de 2017]

Gps, 2017: Oficina de Coordinación Nacional de Posicionamiento, Navegación, y Cronometría por Satélite, Sistema de Posicionamiento Global. <http://www.gps.gov> [Consulta: 30 de Junio de 2017]

Inegi, 2017: Instituto Nacional Mexicano de Estadística y Geografía, Sistema de Posicionamiento Global (GPS) . <http://www.inegi.org.mx/geo/contenidos/geodesia/gps.aspx?dv=c1> [Consulta: 31 de Junio de 2017]

Wikipedia-GLONASS, 2017: The Wikimedia Fondation, Inc., GLONASS. <https://es.wikipedia.org/wiki/GLONASS> [Consulta: 2 de Julio de 2017]

Beidou, 2016: China Satellite Navigation Office , BeiDou Navigation Satellite SystemSignal In SpaceInterface Control Document . <http://www.beidou.gov.cn/attach/2016/11/07/21212.pdf> [Consulta: Consulta: 31 de Junio de 2017]

Gsa, 2017: European GNSS Agency (GSA), Galileo is the European global satellite-based navigation system. <https://www.gsa.europa.eu/european-gnss/galileo/galileo-european-global-satellite-based-navigation-system> [Consulta: 1 de Julio de 2017]

Wikipedia-NFC, 2017: The Wikimedia Fundation, Inc., Near field communication. <> [Consulta: 2 de Julio de 2017]

Kontakt, 2017: Kontakt.io, Your solution. Beacon-enabled. <https://kontakt.io> [Consulta: Consulta: 2 de Julio de 2017]

Chicano, 2016: Chicano, F., Luque, G. y Alba, E.(2016). Smart Cities. Westfort, MA (EE. UU.): Springer International Publishing

Wikipedia-Datos abiertos, 2017: The Wikimedia Foundation, Inc., Datos Abiertos. <https://es.wikipedia.org/wiki/Datos_abiertos> [Consulta: 3 de Julio de 2017]

Madrid, 2017: Ayuntamiento de Madrid, Portal de datos abiertos del ayuntamiento de Madrid. <http://datos.madrid.es> [Consulta: 3 de Julio de 2017]

Barcelona, 2017: Ayuntamiento de Barcelona, Conjuntos de datos - Open Data Barcelona. <http://opendata-ajuntament.barcelona.cat> [Consulta: 3 de Julio de 2017]

Valencia, 2017: Ayuntamiento de valencia, Portal Transparencia y Datos Abiertos Valencia. <http://gobiernoabierto.valencia.es> [Consulta: 3 de Julio de 2017]

IEEE, 1998: The Institute of Electrical and Electronics Engineers, Inc., 830-1998 - IEEE Recommended Practice for Software RequirementsSpecifications, 1998

Booch, 2014: Booch, G., Rumbaugh, J. y Jacobson, I.(2014). The Unified Modeling Language User Guide. Westfort, MA (EE. UU.): Addison-Wesley

Google-location, 2017: Google inc., . <https://developers.google.com/android/reference/com/google/android/gms/location/package-summary> [Consulta: 2 de Septiembre de 2017]

Eclipse-paho, 2017: The Eclipse Foundation, Eclipse Paho - MQTT and MQTT-SN software. <http://www.eclipse.org/paho/> [Consulta: 2 de Septiembre de 2017]

Kontakt-Developer, 2017: Kontakt.io, Kontakt.io Dev Center. <https://developer.kontakt.io> [Consulta: 2 de Septiembre de 2017]

Shneiderman, 2009: Shneiderman, B., Plaisant, C.(2009). Designing the user interface, 5th. Edition. Boston, MA: Addison-Wesley

Esegece, 2017: eSeGeCe, HTML5 Components for Delphi, CBuilder, Lazarus, Firemonkey, C# and Intraweb. <http://www.esegece.com> [Consulta: 3 de Septiembre de 2017]

BOE-A-1999-23750, 1999: Agencia Estatal BOLETÍN OFICIAL DEL ESTADO, BOE.es - Documento BOE-A-1999-23750. <https://www.boe.es/buscar/doc.php?id=BOE-A-1999-23750> [Consulta: 8 de Septiembre de 2017]

BOE-A-2008-979, 2007: Agencia Estatal BOLETÍN OFICIAL DEL ESTADO, BOE.es - Documento consolidado BOE-A-2008-979. <https://www.boe.es/buscar/act.php?id=BOE-A-2008-979> [Consulta: 8 de Septiembre de 2017]

AGPD-Resoluciones, 2017: Agencia Española de Protección de Datos, Agencia Española de Protección de Datos - Resoluciones. <http://www.agpd.es/portalwebAGPD/resoluciones/index-ides-idphp.php> [Consulta: 8 de Septiembre de 2017]

AGPD-Recomendaciones, 2017: Agencia Española de Protección de Datos, Agencia Española de Protección de Datos - Recomendaciones, 2017

BOE-A-2002-13758, 2002: Agencia Estatal BOLETÍN OFICIAL DEL ESTADO, BOE.es - Documento consolidado BOE-A-2002-13758<. <https://www.boe.es/buscar/act.php?id=BOE-A-2002-13758> [Consulta: 8 de Septiembre de 2017]

BOE-A-1994-14121, 2004: Agencia Estatal BOLETÍN OFICIAL DEL ESTADO, BOE.es - Documento BOE-A-1994-14121. <https://www.boe.es/buscar/doc.php?id=BOE-A-1994-14121> [Consulta: 8 de Septiembre de 2017]

7. Acrónimos

Acrónimo	Significado
BDT	BeiDou navigation satellite system Time
BPMN2	Business Process Modeling Language 2
CORBA	Common Object Request Broker Architecture
DCOM	Distributed Component Object Model
GEO	Geostationary Earth Orbit
GNSS	Global Satellite Navigation System
GPS	Global Positioning System
HTML	Hypertext Markup Languaje
IERS	International Earth Rotation and Reference System Service
IGSO	Inclined Geosynchronous Satellite Orbit
JDBC	Java Database Connectivity
JSON	JavaScript Object Notation
LWT	Last Will and Testament
MEO	Medium Earth Orbit
MQTT	Message Queue Telemetry Transport
MQTT-SN	MQTT for Sensor Networks
NFC	Near Field Communication
ODBC	Open Database Connectivity
ORDBMS	Object-Relational Data Base Management System
RDBMS	Relational Database Management System
REST	Representational State Transfer
RF	Requisitos funcionales
RNF	Requisitos no funcionales
RFID	Radio-Frequency Identification
RMI	Remote Method Invocation
SGML	Standard Generalized Markup Languaje
SIP	Sistema de Información Poblacional
SSL	Secure Sockets Layer
TI	Tecnologías de la Información
TLS	Transport Layer Security
TSQL	Transact-SQL
UML	Unified Modeling Language
URL	Universal Resource Locator
UTC	Coordinated Universal Time
YAML	YAML Ain't Markup Language

Anexo I. Configuración de Mosquitto

En el archivo de configuración de Mosquitto se han modificado las siguientes opciones, dejando el resto según los valores por defecto:

```
# Config file for mosquitto
#
# See mosquitto.conf(5) for more information.
#
...
# Choose the protocol to use when listening.
# This can be either mqtt or websockets.
# Websockets support is currently disabled by default at compile time.
# Certificate based TLS may be used with websockets, except that
# only the cafile, certfile, keyfile and ciphers options are supported.
protocol websockets
...
# Save persistent message data to disk (true/false).
# This saves information about all messages, including
# subscriptions, currently in-flight messages and retained
# messages.
# retained_persistence is a synonym for this option.
persistence true

# The filename to use for the persistent database, not including
# the path.
persistence_file mosquitto.db
# Location for persistent database. Must include trailing /
# Default is an empty string (current directory).
# Set to e.g. /var/lib/mosquitto/ if running as a proper service on
Linux or
# similar.
persistence_location /var/lib/mosquitto
...
# Boolean value that determines whether clients that connect # without
providing a username are allowed to connect. If set to
# false then a password file should be created (see the
# password_file option) to control authenticated client access.
# Defaults to true.
allow_anonymous true
```