Universidad del País Vasco
Euskal Herriko Unibertsitatea

Máster y Doctorado en

**S**istemas **I**nformáticos **A**vanzados

Informatika Fakultatea – Facultad de Informática

Tesis de Máster

# Análisis de soluciones innovadoras para desarrollar aplicaciones cliente-servidor con tratamiento avanzado de información *en la nube*

**Ibai Valencia Itoiz**

Director

**José Miguel Blanco Arbe**

Lenguajes y Sistemas Informáticos

Facultad de Informática de Donostia – San Sebastián. UPV-EHU

Máster S.I.A.

informatika fakultatea
facultad de informática

LSI

10 de julio de 2013

# Análisis de soluciones innovadoras para desarrollar aplicaciones cliente-servidor con tratamiento avanzado de información *en la nube*

**Abstract**

*Este trabajo de fin de master realiza un análisis de una serie de tecnologías emergentes relacionadas con el tratamiento avanzado de la información en entornos distribuidos y escalables del lado servidor, el desarrollo de aplicaciones para smartphones multiplataforma no-nativas que hacen uso de la geo-posición del terminal y el desarrollo del software. Para ello, se estudian tecnologías abiertas como Node.js para el servidor, MongoDB y Cassandra como SGBDs NoSQL y varias funcionalidades HTML5 combinadas con frameworks javascript para resolver un problema tipo: un caso práctico de una aplicación móvil cliente-servidor que integrará dichas tecnologías y se albergará en la nube.*

**Keywords**

*Aplicaciones móviles, cliente-servidor, desarrollo no-nativo, HTML5, Node.js, javascript, MongoDB, Cassandra, Jade-lang, IndexedDB, localStorage, geo-posicionamiento, nube.*

Ibai Valencia Itoiz

## Tabla de contenido

# 1. Introducción

El objeto de este trabajo fin de máster es el de analizar la adecuación de una serie de tecnologías emergentes en el ámbito del tratamiento avanzado de información en entornos distribuidos y escalables en el lado del servidor, el desarrollo de aplicaciones para smartphones multiplataforma que hacen uso de la geo-posición del terminal y el desarrollo del software.

El desarrollo de aplicaciones específicas para smartphones ha tenido un fuerte impacto en el mundo del software desde la salida al mercado de los conocidos smartphones táctiles y sus tiendas de aplicaciones. Existen dos orientaciones para la implementación de estos desarrollos: la línea nativa y la línea no nativa.

La motivación de este trabajo parte de una experiencia previa en el desarrollo de una aplicación multiplataforma nativa para smartphones. Sin embargo, en los últimos años está emergiendo un fuerte interés por el desarrollo de aplicaciones móviles no-nativas de orientación web [sind2012] [gawr2012] por su carácter de multi-plataformidad y la gran compatibilidad con todo tipo de dispositivos: desde móviles, tabletas y escritorios hasta video-consolas o aplicaciones de televisión.

Este es el motivo por el que se quiere realizar un estudio de las herramientas y tecnologías emergentes para experimentar si con las mismas se pueden lograr resultados comparables a los conseguidos con la solución nativa.

Este trabajo estudiará una plataforma de servidor preparada para sistemas de alto rendimiento, distribuidas y escalado como Node.js, analizará Cassandra y MongoDB como sistemas de gestión de bases de datos NoSQL y probará la adecuación y factibilidad de utilizar el estándar HTML5 combinado con CSS, javascript y el framework para aplicaciones móviles jQuery Mobile para el desarrollo de aplicaciones móviles.

Posteriormente, se tratará de integrar las tecnologías y herramientas estudiadas en la implementación de una prueba de concepto cliente-servidor con las características mencionadas anteriormente y que se albergará en un servidor Cloud como el servicio Amazon Web Services.

El ámbito de ésta investigación se concreta en las áreas y contenidos vistos durante el curso del máster en Sistemas Informáticos Avanzados relacionados con el Desarrollo del Software Seguro en Entornos Web, la Computación en Red al enfocar la solución al entorno de la nube, el Desarrollo de Sistemas Innovadores de Gestión de Datos considerando dos SGBD orientados al NoSQL y Sistemas Ubicuos al devolver información relacionada con el entorno, entre otros.

Para situar la discusión en contexto, el trabajo comienza analizando la historia reciente de la WWW desde sus inicios en 2. Evolución de la tecnología y continúa introduciendo en el 3. Estado de la cuestión, la actualidad respecto al desarrollo de aplicaciones web y el futuro próximo con las nuevas plataformas planteadas para este curso y las tendencias en el desarrollo de aplicaciones del lado del servidor.

En el apartado 4. Descripción de la prueba de concepto, se planteará el desarrollo de una aplicación tipo sobre la que se probará la solución más adecuada y que integrará las tecnologías y arquitecturas para una aplicación cliente-servidor presentadas en los siguientes apartados: 5. Tecnologías y 6. Arquitecturas. Dichos apartados propondrán diferentes alternativas tecnológicas para la implementación de una aplicación cliente-servidor y se describirá un alcance ambicioso para poder plantear una arquitectura adecuada para la prueba de concepto.

Tanto en 7. Servidor como en 8. Cliente se presentan el funcionamiento, el diseño y el desarrollo de la aplicación, profundizando en la arquitectura y en el uso de las tecnologías escogidas para dar solución al problema planteado y explicar cómo utilizarlas e implementarlas.

Por último, el trabajo termina con unas conclusiones personales en el apartado 9. Conclusiones, que engloban y resumen los resultados experimentados durante la implementación del caso de prueba y las tecnologías escogidas, y en 10. Trabajo futuro, se presenta el trabajo que queda abierto y se estudiaría en una ocasión futura para continuar la línea en el ámbito que se ha explorado.

# 2. Antecedentes

Este apartado contiene un análisis de los hitos más importantes de la historia de internet, desde los inicios de la World Wide Web en 1989 hasta los comienzos y asentamiento de las plataformas móviles en 2011.

Posteriormente, incorpora información de las arquitecturas y funcionamiento de las plataformas móviles más utilizadas y afincadas a día de hoy: Apple iOS y Android; y termina con una experiencia personal y motivación del proyecto fin de máster.

## 2.1. Los comienzos de internet

Apenas han pasado veinticuatro años desde que Tim Berners-Lee propusiera en 1989 un cambio en la forma de funcionar de internet [bert1989] [weaa1998], utilizando un nuevo concepto conocido como *hypertext* que, combinado con el protocolo HTTP y las URL, revolucionaría el futuro inmediato de las comunicaciones. Su primer nombre sería Mesh y evolucionaría a World Wide Web en 1990.

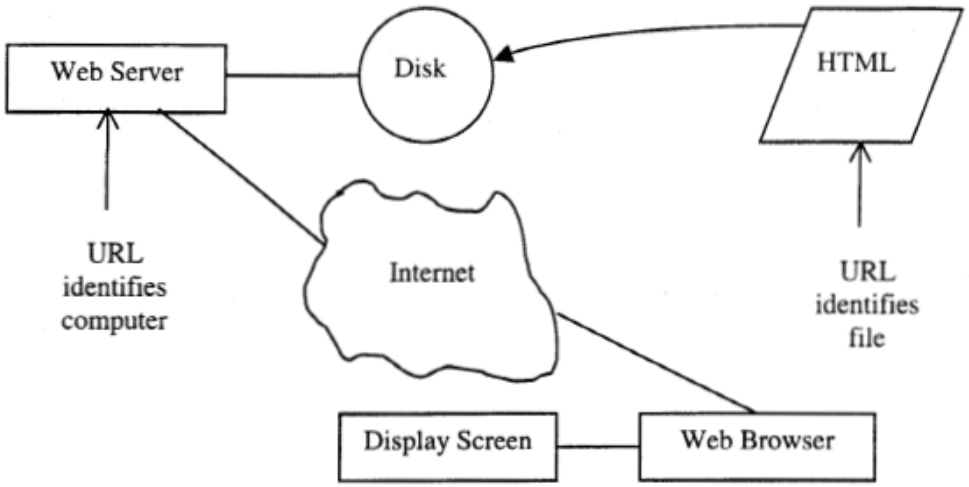



**Ilustración 1 - Elementos de la WWW [weaa1998]**

La primera versión de HTTP conocida como HTTP v0.9 fue documentada oficialmente en 1991[1] y únicamente contaba con un verbo: GET, al que se le hacía una solicitud a través de una URL y devolvía un mensaje de lenguaje de marcado

[1] HTTP v0.9: http://www.w3.org/Protocols/HTTP/AsImplemented.html (apr 26, 2013)

hipertextual, HTML. El protocolo siempre devolvía un contenido HTML, los mensajes no se tipaban por lo que, a excepción de la lectura del contenido del mensaje, no era posible identificar un error.

Para consumir un documento HTML, WorldWideWeb contaba también con una aplicación que era utilizada como navegador y que fue renombrada a Nexus para separar las dos capas convirtiéndose en el primer navegador web.

Ese mismo año, se constituye una organización no gubernamental y sin ánimo de lucro conocida como Internet Society[2] (ISOC), que se dedicará a desarrollar y dar soporte a protocolos y estándares relacionados con el nuevo concepto de Internet.

En 1993, el código de la WorldWideWeb fue publicado como proyecto de dominio público[3] para asegurar un mayor soporte corporativo provocando que surgiera un nuevo navegador [groa2005]: el Mosaic. Mosaic[4] fue desarrollado por un grupo de la National Center for Supercomputing Applications de la Universidad de Illinois en 1993 y terminaría teniendo versiones para Mac OS, Windows y AmigaOS.

Un año más tarde, en 1994, aparece otra asociación conocida como la World Wide Web Consortium (W3C)[5] encabezada por el propio Tim Berners-Lee en el laboratorio de la Computer Science en el MIT con el objetivo de evolucionar los protocolos y estándares asociados a la Web.

[2] Historia de internet: http://www.internetsociety.org/internet/what-internet/history-internet/brief-history-internet (apr 23, 2013)
[3] CERN.ch: WWW de dominio publico http://tenyears-www.web.cern.ch/tenyears-www/Welcome.html (jun 22, 2013)
[4] Mosaic: http://www.ncsa.illinois.edu/Projects/mosaic/history.html (apr 24, 2013)
[5] W3C: Inicios de la W3C http://www.w3.org/Consortium/facts.html#history (jun 23, 2013)

## 2.2. El éxito de Internet

Netscape Navigator fue desarrollado junto con extrabajadores [groa2005] de Mosaic, y se convertiría en el navegador que mejor se posicionaría con la explosión de internet en 1994.

El Internet Explorer, de Microsoft, fue lanzado solo un año más tarde y comenzaría a pisarle los talones a Netscape Navigator. Gracias a ser el navegador por defecto de Windows, a las campañas de publicidad y a que era compatible con Macintosh[6], no tardó en convertirse en el navegador más exitoso de la década.

| Month | Netscape Navigator | Microsoft Internet Explorer |
|---|---|---|
| May-96 | 83.2% | 7.0% |
| Jun-96 | 78.2% | 8.3% |
| July-96 | 72.6% | 15.8% |
| Aug-96 | 62.7% | 29.1% |

**Ilustración 2 - Porcentaje del mercado de los navegadores *Intersé Corp.**

Poco después, se actualiza el protocolo HTTP a la versión 1.0 que, tal y como aclara la especificación[7], además del verbo GET, incorporaría HEAD y POST. La popularidad de internet continúa en pleno auge, empezaron a surgir sitios web conocidos mundialmente a día de hoy como Amazon, Yahoo, eBay y MSN en 1995, Google (1998) o Wikipedia (2001). MySQL[8] ve la luz en 1998 para Windows 98 y NT con su primera versión de producción.

[6] Microsoft: Available on all Major Platforms http://www.w3.org/Consortium/facts.html#history (jun 23, 2013)
[7] HTTP 1.0, 1996: http://www.w3.org/Protocols/HTTP/1.0/draft-ietf-http-spec.html (apr 25, 2013)
[8] MySQL, 1998: http://www.mysql.com/about/ (apr 26, 2013)

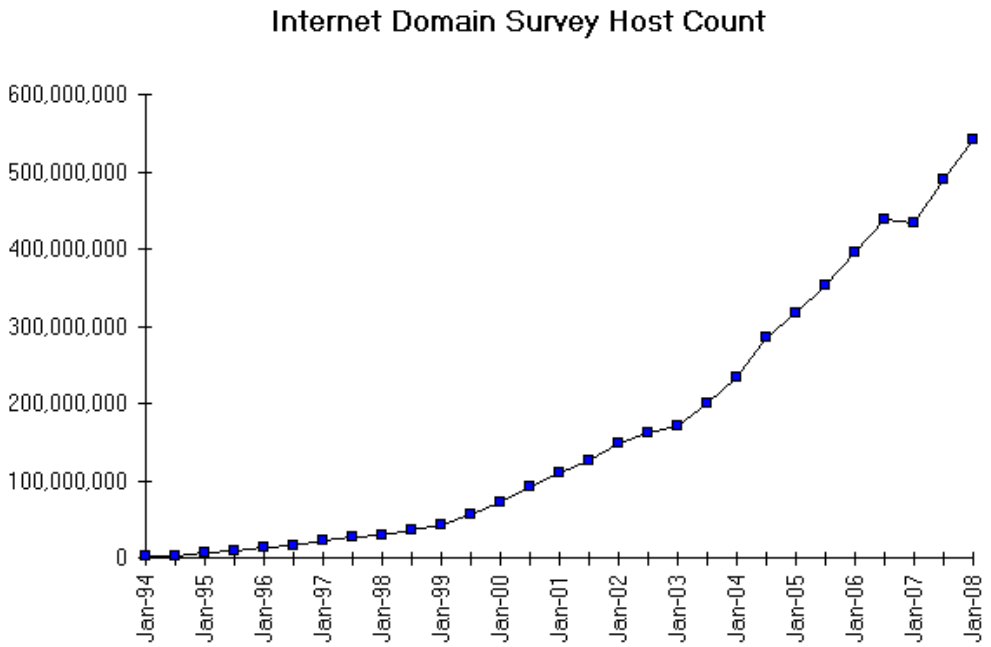



**Ilustración 3 – Crecimiento del número de hosts por año *www.isc.org[9]**

En 1999 surge el primer navegador que comenzaría a discutir el monopolio de Internet Explorer: Mozilla. Además, en este año IEEE oficializa el estándar 802.11[10], conocido comúnmente como Wifi, encargado de las conexiones inalámbricas para acceder a internet; se propone una actualización de la especificación de HTML, versión 4.01[11]; y se oficializa la versión 1.1[12] de HTTP en la que colaboran diferentes grandes compañías, entre otras, la W3C con T. Berners-Lee, Microsoft, Xerox o Compaq. Esta versión ya incorporaría en la especificación todos los verbos conocidos hasta la fecha.

En 2003, comienzan a aparecer alternativas al Internet Explorer, se funda Mozilla Foundation[13], que lanzaría Firefox apenas un año más tarde y Apple saca[14] al mercado el que sería el navegador por defecto de Macintosh, Safari.

[9] ISC: Hosts survey https://www.isc.org/services/survey/ (jun 23, 2013)
[10] IEEE 802.11, Wifi. 1999: http://www.ieee802.org/11/ (apr 26, 2013)
[11] W3: HTML 4.01 1999: http://www.w3.org/TR/REC-html40/ (apr 29, 2013)
[12] W3: HTTP 1.1 1999: http://www.w3.org/Protocols/rfc2616/rfc2616.html (apr 29, 2013)
[13] Mozilla: History http://www.mozilla.org/en-US/about/history/ (apr 29, 2013)
[14] Apple: Apple Releases Safari http://www.apple.com/pr/library/2003/06/23Apple-Releases-Safari-1-0.html (jun 23, 2013)

## 2.3. La web 2.0 y el internet moderno

Durante los siguientes años, empiezan a darse los primeros pasos de la web de las comunicaciones e interacciones entre personas, conocida como web 2.0 cuyas características, según Tim O'Reilly, fundador de O'Reilly Media y conocido como uno de los autores de la Web 2.0, son "seeing the Web as a platform, harnessing collective intelligence, being data-centered, using Web-based applications, using lightweight programming models, using cross-device applications, and providing rich user experiences"[15].

En ésta época se fundan y dan a conocer las siguientes webs: LinkedIn, MySpace, Orkut (Google), Facebook, Youtube y, unos años después, Twitter y Tumblr. Google, además, lanza nuevos servicios y los adapta a los primeros móviles compatibles con la web: Local, Blogger, su motor de búsquedas y Gmail.

Apple, Mozilla Foundation y Opera Software, visto que el rumbo que llevaba la W3C se dirigía hacia el XHTML dejando de lado al HTML y las necesidades reales, decidieron fundar la WHATWG[16] (Web Hypertext Application Technology Working Group) en 2004, que centra su esfuerzo en evolucionar el mundo web y crear APIs para las aplicaciones web. Este mismo año se lanza una nueva distribución de Linux conocida como Ubuntu[17] y llevada por la compañía Canonical, constituida para dar soporte al sistema operativo.

La rápida evolución de la web hace que se comiencen a ver determinadas limitaciones a la hora de crear aplicaciones webs más ricas en contenidos, accesibles y semánticas. Por ello, en 2006, la W3C y la WHATWG deciden comenzar una cooperación [juna2013] y empiezan a plantear una nueva versión de HTML conocida como HTML5[18] cuyas especificaciones empiezan a ser implementadas en los navegadores más modernos pero que no se estandarizará hasta 2014. Se hablará más sobre HTML5 en el apartado de 4. Estado de la cuestión, 5. Tecnologías y 8. Cliente.

[15] O'Reilly, la web 2.0 http://www.oreillynet.com/lpt/a/6228 (jun 23, 2013)
[16] WHATWG, 2004: http://wiki.whatwg.org/wiki/FAQ (apr 29, 2013)
[17] Ubuntu: La historia http://www.ubuntu.com/about/about-ubuntu (jun 23, 2013)
[18] HTML5 especificaciones: http://www.w3.org/html/wg/drafts/html/master/ (apr 28, 2013)

## 2.4. Los comienzos del Smartphone

2007 fue un año importante y supuso un gran cambio en la forma de consumir la informática y la tecnología. Este año Apple sorprendió al mundo con el primer Smartphone con pantalla táctil, el iPhone[19]; y Google anunció, en forma de respuesta, un sistema operativo para móviles conocido como Android que sería implementando en el primer[20] smartphone en 2008 con un HTC Dream.

Estos teléfonos revolucionarios suponen un gran cambio a la hora de utilizar internet. Incorporan antenas Wifi y ponen de moda las tarifas de consumo de datos que permiten navegar por internet cuando no se dispone de señal Wifi.

Se genera un nuevo área de desarrollo, una nueva forma de consumir, donde los desarrolladores crean aplicaciones nativas para estas nuevas plataformas y se añaden a un revolucionario nuevo concepto para vender las aplicaciones [ghea2010]: las tiendas de aplicaciones oficiales donde los clientes las descargan de forma gratuita o de pago.

Los navegadores de los smartphones están preparados para escalar las páginas web y mostrarlas de forma adecuada en pantalla. Sin embargo, surge un nuevo movimiento para adaptar las páginas web a las pantallas de los teléfonos y, por este motivo, en 2008 la W3C decide publicar unas prácticas recomendadas[21] para desarrollar aplicaciones en las que se incluye el concepto *responsive*[22], que consiste en adaptar la visualización de las páginas web al navegador del dispositivo que se utiliza.

Para ésta época, el protocolo de direcciones IPv4[23] descrito en 1981, comienza a agotar sus reservas de direcciones IP y se lanza y comienza a usar oficialmente su

[19] iPhone, 2007: http://www.apple.com/pr/products/iphone/iphone.html (apr 29, 2013)
[20] HTC Dream (G1) con Android: http://web.archive.org/web/20110712230204/http://www.htc.com/www/press.aspx?id=66338&lang=1033 (jun 23, 2013)
[21] W3C: Mobile web best practices http://www.w3.org/TR/mobile-bp/ (apr 29, 2013)
[22] W3C: One web, diseño adaptativo a móviles http://www.w3.org/TR/mobile-bp/#OneWeb (jun 23, 2013)
[23] IPv4, 1981: http://www.ietf.org/rfc/rfc791.txt (apr 29, 2013)

sucesor, IPv6[24], propuesta en 1998, cuya longitud de dirección es de 128 bits permitiendo $2^{128}$ direcciones.

En 2009, CISCO revela que el tráfico de datos de los nuevos teléfonos móviles supera[25] al tráfico de voz por primera vez, este dato también es apoyado por Ericsson[26]. Lo que refleja el claro éxito de los smartphone y su prometedor futuro.

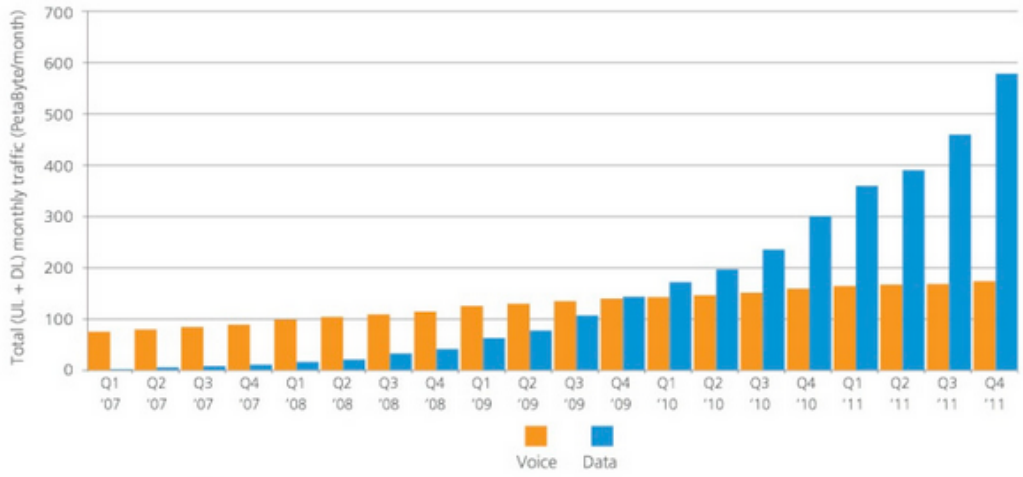



**Ilustración 4 - Comparativa tráfico voz y datos. Datos recogidos por Ericsson**

Un año más tarde se presenta un nuevo tipo de dispositivo que continuaría la revolución del consumo de internet: los *tablet*. El iPad, de Apple, fue el primer dispositivo[27] de esta clase, aprovechaba el sistema operativo del iPhone adaptando la interfaz a este nuevo formato. Poco después Android seguiría sus pasos adaptando su plataforma y sacando al mercado, de la mano de Motorola, el Tablet Xoom.

Microsoft se unía tarde al mercado de los teléfonos actualizando[28] su obsoleto sistema operativo de móviles a Windows Phone 7 y sacaría en 2012 al mercado la primera Tablet con Windows 8.

[24] IPv6, 1998: http://tools.ietf.org/html/rfc2460 (apr 29, 2013)
[25] CISCO: Tráfico de datos supera a voz http://www.cisco.com/en/US/solutions/collateral/ns341/ns973/ns1114/white_paper_c11-676683.html (apr 29, 2013)
[26] Ericsson: Tráfico de datos supera a voz http://www.ericsson.com/thecompany/press/releases/2010/03/1396928 (jun 29, 2013)
[27] Apple: Launches iPad http://www.apple.com/pr/library/2010/01/27Apple-Launches-iPad.html (jun 29, 2013)
[28] Microsoft: Windows 7 News http://www.microsoft.com/en-us/news/features/2010/oct10/10-11WP7main.aspx (23 jun, 2013)

## 2.5. Plataformas móviles: Apple iOS y Android

Desde la salida de los sistemas operativos para móviles iOS y Android en 2007-08 y su orientación a las pantallas táctiles, no han parado de crecer, desbancando a la competencia existente en poco tiempo y haciéndose con el mercado móvil.

En la gráfica adjunta, se puede ver el estudio realizado por la compañía comScore Inc. que muestra el estado de suscripciones de cada plataforma en Marzo de 2013.

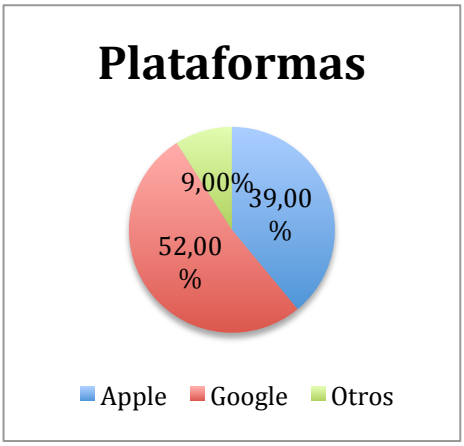



**Ilustración 5 - Gráfica de suscripción de plataformas móviles *comScore Inc.**

### Apple iOS

El iPhone, presentado por Apple en 2007, fue el primer dispositivo móvil táctil con las características propias de un smartphone. Según la documentación [29] de Apple, la plataforma, conocida como iOS, ejecuta código nativo del procesador compilado desde Objective-C, el lenguaje que utilizan los desarrolladores de iOS y, debido a que Apple fabrica sus propios dispositivos, no ha tenido el problema de tener que ejecutarse en diferentes procesadores, con lo que su fabricación ha sido ajustada a las características del hardware para conseguir el mejor resultado.

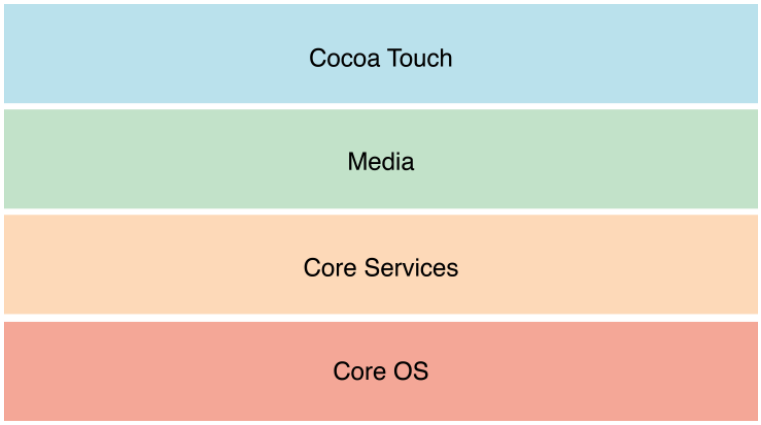



**Ilustración 6 - Arquitectura iOS *developer.apple.com**

[29] Apple: iOS Technology https://developer.apple.com/library/ios/#documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html#//apple_ref/doc/uid/TP40007898 (jun 23, 2013)

La arquitectura de iOS está formada por cuatro capas[30]: la capa Core OS, basada en UNIX, es la más baja y se sitúa sobre el hardware; la Core Services contiene los servicios fundamentales del sistema que utilizan todas las aplicaciones; la capa Media es la encargada de proporcionar las tecnologías de gráficos, audio y vídeo; y, la capa de más alto nivel Cocoa Touch, contiene los frameworks necesarios para desarrollar una aplicación iOS.

Como todos los dispositivos de la compañía, Apple es la encargada de generar hardware y la plataforma sobre la que se ejecutan las aplicaciones de los desarrolladores. Por tanto, son capaces de optimizar su funcionamiento y hacer que las aplicaciones aprovechen toda la potencia del dispositivo.

Además, gracias al sistema de actualizaciones que utilizan y el control de los dispositivos a la venta en el mercado, logran que la fragmentación sea mínima y prácticamente todo dispositivo tenga instalada la última versión.

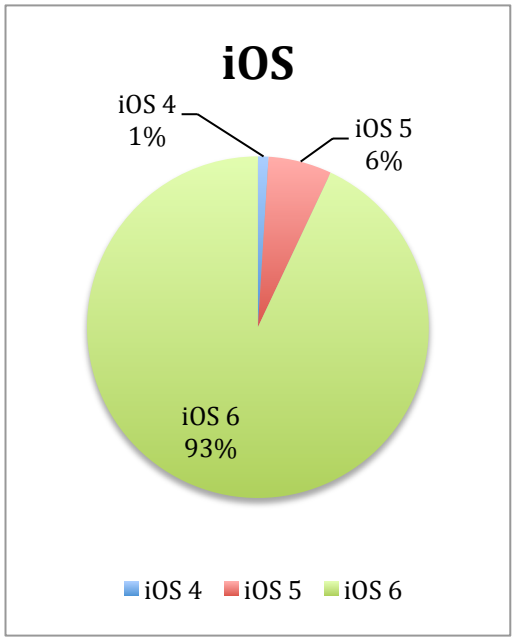



**Ilustración 7 - Datos de la WWDC13 de Apple**

Sin embargo, algo de lo que siempre se le ha criticado a Apple ha sido su estricto, y en ocasiones abusivo, control de las aplicaciones que se publican en la Apple Store. No existe otra vía oficial de distribuir aplicaciones para iOS y la flexibilidad para realizar aplicaciones de uso concreto a veces es limitada. Por ejemplo, no permiten que las aplicaciones se ejecuten en segundo plano excepto si lo hacen de la forma que aparece en la

[30] Capas de iOS: https://developer.apple.com/library/ios/#documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html (apr 28, 2013)

documentación oficial[31]. Sin embargo, según la keynote WWDC2013, la política de ejecución de aplicaciones en segundo plano es algo que cambiará con iOS7.

### Android

El desarrollo del sistema operativo Android fue comenzado por Android Inc. en 2003 hasta que fue adquirido por Google en 2005 y en 2007 se creó la Open Handset Alliance para impulsar su desarrollo [trak2012].

La arquitectura[32] de Android está basada en cuatro capas: sobre el hardware monta un kernel Linux, seguido de una capa de software con librerías para acceder al hardware. El *runtime* de Android es una máquina virtual de java optimizada para móviles, conocida como Dalvik, sobre la que se ejecutarán las aplicaciones, y una capa *framework* donde se implementan las APIs del sistema a las que los desarrolladores tienen acceso.

Por tanto, las aplicaciones desarrolladas para esta plataforma están escritas en Java, lo que ha favorecido y ha sido uno de los motivos de su enorme éxito por ser un entorno flexible y muy conocido con el que prácticamente se puede desarrollar cualquier solución sin problemas de ser rechazada por la Google Play.

Sin embargo, el que la plataforma deba de utilizar una máquina virtual Java ha sido uno de los motivos más criticados por la comunidad de desarrolladores, ya que penaliza mucho el uso de los recursos y la potencia del dispositivo.

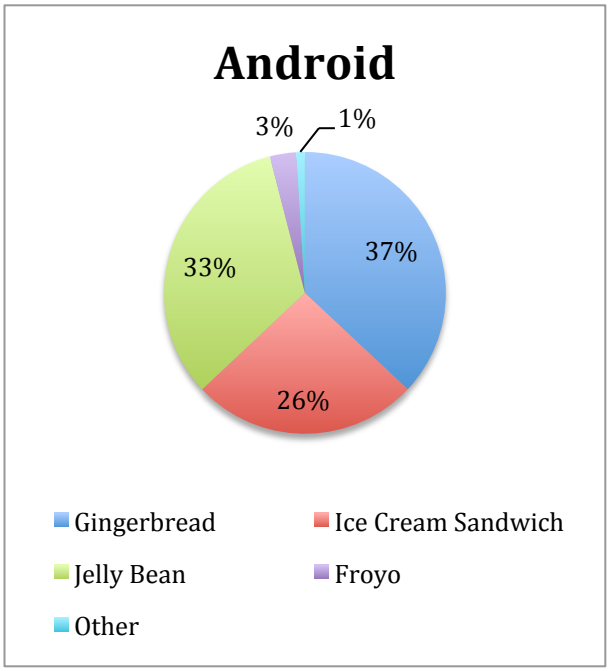



**Ilustración 8 - Datos del Dashboard de Android Developer y WWDC13**

[31] Apple: Tipos de ejecución en segundo plano http://developer.apple.com/library/ios/#documentation/iphone/conceptual/iphoneosprogrammingguide/ManagingYourApplicationsFlow/ManagingYourApplicationsFlow.html (jun 23, 2013)
[32] Arquitectura de Android: http://developer.android.com/about/versions/index.html (apr 28, 2013)

Un factor que se está notando mucho con los años es la fragmentación a la que se enfrenta Android y es algo que se puede observar en la ilustración 8.

Se está convirtiendo en un serio problema para Google haciendo que las versiones existentes en los diferentes dispositivos sean descontinuadas porque cada fabricante de smartphones personaliza la plataforma y después no actualizan. Por tanto, los desarrolladores han de hacer un gran esfuerzo por mantener la compatibilidad de sus aplicaciones en todas las versiones de Android y en todos los dispositivos con hardware y pantallas diferentes.

## 2.6. Motivación y experiencia personal

Queda argumentado que, actualmente, el mercado móvil es un mercado muy demandado por las empresas, los clientes y las tendencias. Las posibilidades son grandes y se augura un futuro prometedor y amplio en el uso y el desarrollo para estos dispositivos.

Además, como se introducirá en el siguiente apartado Estado de la cuestión, ya están surgiendo nuevas alternativas y tendencias para el desarrollo de aplicaciones de smartphones y tablets en las que se basan en soluciones open-source como HTML5.

Mi interés personal hacia el desarrollo orientado a plataformas móviles comienza en 2011 cuando, junto a dos compañeros de carrera, decidimos centrar nuestro proyecto fin de carrera en el desarrollo de una aplicación móvil cliente-servidor que hiciera uso de el dispositivo GPS del smartphone.

La aplicación experimentaba con prácticamente todas las APIs de los dispositivos. Multi-plataforma, desarrollada y publicada en las Stores de Apple iOS y Android, y desarrollada para Windows Phone, hace uso del servicio de localización y segundo plano de las plataformas, utiliza las tecnologías PUSH que ponen a disposición sus servidores específicos, hace uso del servicio de mapas, establece relaciones (basadas en localización) entre amistades por medio un servidor intermediario,

recorre la agenda de contactos con el fin de localizar amistades, permite a los usuarios establecer y configurar su grado de privacidad, hace uso de sistemas de mensajería para verificar el perfil de un usuario y un largo etc. que se puede completar leyendo la memoria GeoFriends de Ingeniería Informática en la facultad FISS de la UPV-EHU [alme2012] [gari2012] [vali2012] o probando la propia aplicación GeoFriends[33] disponible en la Apple Store y Google Play.

El interés en el desarrollo de aplicaciones para dispositivos móviles sigue vigente y, en esta ocasión, se pretende investigar sobre las alternativas al desarrollo de software nativo, experimentar con entornos abiertos como el estándar HTML5 combinado con la tecnología JavaScript y CSS para el desarrollo de aplicaciones web que puedan ejecutarse en cualquier plataforma móvil.

[33] GeoFriends: http://www.geofriendsapp.com (apr 29, 2013)

# 3. Estado de la cuestión

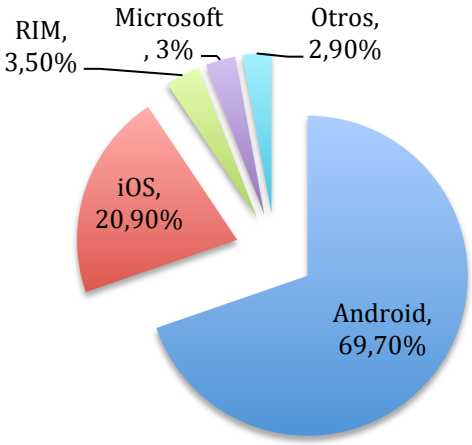
En este apartado se va a presentar la actualidad de las tecnologías que tienen relación con el trabajo que se ha realizado. Se comienza haciendo una introducción a los sistemas operativos móviles asentados actualmente y los recientemente anunciados, se hablará del concepto *cloud*, las tendencias y por último, de los sistemas de gestión de bases de datos NoSQL.

## 3.1. Plataformas móviles actuales

Si bien el mundo de los smartphones es relativamente nuevo y ocupa el debate de actualidad, como se ha mencionado anteriormente, ya lleva unos años de recorrido y está estabilizado desde que Apple revolucionó el mercado de la telefonía móvil en 2007 con la salida al mercado del primer iPhone. Desde entonces, el smartphone se ha convertido ya en una herramienta básica en el día a día de las personas [trak2012].

Sin embargo, en los últimos dos años, el uso de internet, los servicios y las tecnologías han evolucionado de manera considerable provocando nuevos debates y tendencias de uso para dar solución a las demandas del mercado.

Según un estudio [34] publicado por la empresa consultora estadounidense Gartner Inc. en Febrero de 2013, en 2012 se vendieron un total de 1.7 millones de teléfonos móviles de los cuales, un 69.7% montaban el sistema operativo Android y un 20.9% iOS; lo que ocupa el 90% del mercado, dejando así en un 3.5% BlackBerry de RIM y en un 3% el sistema operativo de Microsoft.




[34] Smartphones market share: http://www.gartner.com/newsroom/id/2335616 (apr 26, 2013)

Antero Junten [juna2013] afirma que el motivo de compra de un Smartphone, ha cambiado, pasando de valorarse las prestaciones y capacidades hardware que éste dispone, a considerarse las aplicaciones que es capaz de ejecutar. La importancia de las aplicaciones y su importante crecimiento es, en gran parte, debido a la aparición de las tiendas de aplicaciones de las plataformas, ya que han logrado simplificar el proceso de búsqueda e instalación de las mismas haciendo que incremente la demanda.

Además, esta política ha hecho que los pequeños desarrolladores accedan más fácil a la masa de clientes finales y que puedan competir con más facilidades frente a los grandes fabricantes [ghea2010]. Sin embargo, las tiendas de aplicaciones están preparadas para aplicaciones nativas que se descargan al terminal y se ejecutan localmente.

## 3.2. Nuevas plataformas móviles

Al margen de los sistemas operativos móviles dominantes mencionados anteriormente, a comienzos de 2013 se anunciaron tres nuevas plataformas que podrían dar que hablar en los próximos meses. Se trata de Tizen[35], sistema operativo que sacará Samsung para sus teléfonos de gama media-baja; Ubuntu for phones[36] de Canonical, basado en Linux; y la iniciativa de la Fundación Mozilla denominada FirefoxOS[37] que basará la plataforma en el desarrollo de aplicaciones HTML5.

Tanto Ubuntu for phones como FirefoxOS toman como base el trabajo realizado en Android, cuya plataforma es open-source: ambos están montados sobre un núcleo Linux y utilizan herramientas muy similares a las de Android. Sin embargo, dejan de lado la capa Java/Dalvik, lo que les hace diferentes.

### Ubuntu for phones

Como se puede leer en la documentación oficial de esta plataforma, Ubuntu for phones es un sistema muy similar a Android. Elimina la última capa de su arquitectura, en la que se encontraba Java, y basa su desarrollo en el software

[35] Tizen: https://www.tizen.org/ (apr 27, 2013)
[36] Ubuntu for phones: http://www.ubuntu.com/phone (apr 28, 2013)
[37] FirefoxOS: http://www.mozilla.org/en-US/firefoxos/ (apr 28, 2013)

nativo. Eliminado la capa de Java, logra que las aplicaciones se compilen y corran directamente sobre el hardware con lo que es ideal para teléfonos móviles con poca potencia y prestaciones. Pero además, no solo se limita a las aplicaciones nativas, sino que también da soporte completo a aplicaciones basadas en HTML5[38].

El SDK de la plataforma está siendo continuamente mejorado y actualizado y siempre está disponible para su descarga de forma abierta en la web. Esto le diferencia del resto de plataformas móviles, ya que es posible tener las últimas herramientas en tiempo real.

Los desarrolladores pueden programar en JavaScript, C y C++ y utilizar OpenGL de forma nativa para crear aplicaciones que consuman gráficos de forma intensiva. Usa el lenguaje declarativo QML[39] para describir, tanto el diseño como sus acciones y las interfaces de usuario de las aplicaciones. Este lenguaje está basado en JavaScript y aplicarle un diseño es muy similar a utilizar CSS.

La plataforma no comienza de cero, ya que cuenta con mucho software desarrollado para Ubuntu (desktop) y que se aprovecha para la versión móvil. Por ejemplo, las aplicaciones se publican en el Ubuntu Software Centre.
Destacan, además, que una aplicación de escritorio puede ser adaptada[40] para funcionar en Ubuntu phone y viceversa de una forma muy sencilla.

También pone a disposición del desarrollador la potencia del servicio cloud Ubuntu One[41] para almacenar información y datos de las aplicaciones personales de cada usuario, dando soporte para su API.

[38] Ubuntu: desarrollo HTML5 http://insights.ubuntu.com/mobile/developers-get-the-full-support-they-need/ (apr 29, 2013)
[39] QML: http://qt-project.org/doc/qt-4.8/qdeclarativeintroduction.html (apr 28, 2013)
[40] Ubuntu: app escritorio y móvil http://www.ubuntu.com/phone/app-ecosystem (jun 24, 2013)
[41] Ubuntu: Ubuntu One API http://www.ubuntu.com/phone/app-ecosystem (jun 24, 2013)

### Firefox OS

Este sistema operativo móvil impulsado por Mozilla Foundation y desarrollado por su proyecto Boot to Gecko (B2G), también bajo la licencia de software libre, está basado en un núcleo de Linux. En su capa más alta parte de un motor de tiempo de ejecución basado en Gecko, que ejecuta aplicaciones desarrolladas en HTML5, JavaScript y APIs de aplicaciones web[42].

Todo el acceso al hardware está implementado por Gecko[43] y se hace a través de Web APIs estándares. Además, también es la capa encargada de leer el contenido web de las aplicaciones y renderizarlos en pantalla.

Sobre la capa de Gecko, cuando no hay ninguna aplicación de terceros en ejecución, está la capa de la interfaz de usuario de Firefox OS conocida como Gaia[44], que no es más que una aplicación web. Gaia está escrita completamente en HTML, CSS y JavaScript, y es la encargada de dibujar en la pantalla todo lo que aparece después de arrancar Firefox OS: la pantalla de inicio y aplicaciones por defecto.

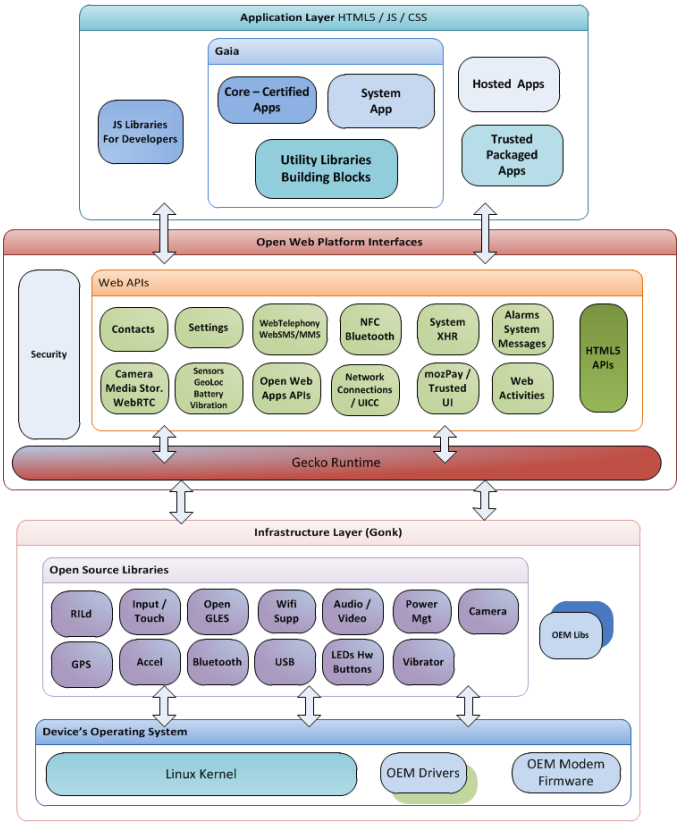



**Ilustración 9 - Arquitectura de Firefox OS**
***developer.mozilla.org**

[42] Firefox OS: https://developer.mozilla.org/es/docs/Mozilla/Firefox_OS (apr 28, 2013)
[43] Gecko: https://developer.mozilla.org/en-US/docs/Mozilla/Gecko (apr, 28, 2013)
[44] Gaia: https://developer.mozilla.org/en-US/docs/Mozilla/Firefox_OS/Platform/Gaia (apr 29, 2013)

Toda aplicación que se desarrolle para el sistema es una aplicación web lanzada por Gaia, y será capaz de acceder al hardware y los servicios del teléfono directamente vía JavaScript a través de Gecko.

### 3.3. Conclusiones línea de desarrollo nativo vs no-nativo

Con las plataformas introducidas anteriormente se abre un nuevo debate, las plataformas existentes hasta día de hoy: Android, iOS, Windows Phone, Symbian, etc. chocarán con los nuevos sistemas operativos orientados a los estándares abiertos que comienzan a desplegarse.

Las plataformas mencionadas anteriormente fueron enfocadas al software nativo, sus aplicaciones se tienen que compilar para cada una de ellas y, por tanto, llevar desarrollos independientes en cada entorno de desarrollo de cada sistema operativo móvil. En cambio, tanto la plataforma Ubuntu for phones como la opción de Mozilla, Firefox OS, dan la posibilidad de desarrollar aplicaciones basadas en estándares web (HTML5). Esto posibilita que un desarrollo para Firefox OS sea completamente compatible con la plataforma de Canonical y, por tanto, sólo sea necesario realizar una vez el trabajo.

La principal ventaja de las aplicaciones nativas frente a las no-nativas, es su rendimiento, los motores de HTML y JavaScript aún no son tan potentes como para poder ponerse a la altura de una aplicación compilada para una arquitectura concreta. Por tanto, una aplicación compleja que necesite la potencia y velocidad del procesador, como son los juegos gráficos, funcionarán mejor si el desarrollo es nativo. En cualquier caso, se sigue trabajando para que los estándares webs puedan hacer un uso más eficaz del hardware y ser comparados con la potencia del desarrollo nativo.

Podemos avanzar que, la nueva tendencia de estas recién anunciadas plataformas, traerá otras ventajas para el resto de aplicaciones y, sobre todo, a los desarrolladores y al bolsillo de los clientes. Principalmente, una aplicación basada en HTML5 y JavaScript es, de por sí, una aplicación multiplataforma que se podrá ejecutar en cualquier sistema operativo (móvil y de escritorio) que disponga de un

navegador. Y además, se reduce el software y librerías necesarias para su ejecución, por tanto, se ahorra en memoria, lo que hace que el hardware sea más barato y que los terminales bajen de precio.

## 3.4. Tendencias para el lado servidor

Contenidos geo-posicionados, redes sociales, consumo masivo de datos, interacción web en tiempo real y juegos multi-jugador en tiempo real, son a día de hoy algunos de los usos más comunes a la hora de utilizar internet, según un estudio[45] publicado por Mashable en abril de 2012. Por este motivo, la arquitectura de las aplicaciones ha evolucionado en los últimos tiempos para poder atender a la demanda tan potente del mercado.

### Cloud Computing

El cloud computing es un concepto relativamente nuevo que cada vez está siendo más utilizado para dar solución a los problemas y tendencias de la actualidad. La US National Institute of Standards and Technology (NIST) define el Cloud computing como: "On-demand self-service, Ubiquitous network access, Resource pooling, Rapid elasticity (resources can be scaled up and down easily), Metered service (resources' usage is measured) and Pay-as-you-Consume business models" [kalp2011]. Por tanto, para las empresas es una solución a tener en cuenta por la escalabilidad horizontal que ofrece. Esto hace que únicamente se tenga que pagar por el uso real que se le de ya que, en momentos puntuales, cuando existen picos de uso, se añaden más máquinas al sistema de forma transparente y ubicua, y se escala para adaptarse al alto consumo puntual. Gracias a esto, se evita tener que realizar migraciones costosas a máquinas más potentes cuando el hardware se queda pequeño o tener que haces estudios a futuro de la infraestructura que será necesaria para los desarrollos. También se evita pagar por hardware que no está en uso cuando la demanda del sistema es menor de lo contratado.

[45] Mashable: Hot media trends: http://mashable.com/2012/04/19/hot-media-trends/ (apr 28, 2013)

En la siguiente ilustración se observa un extracto de un diagrama que describe el concepto del cloud computing extraído de Cloud Patterns.

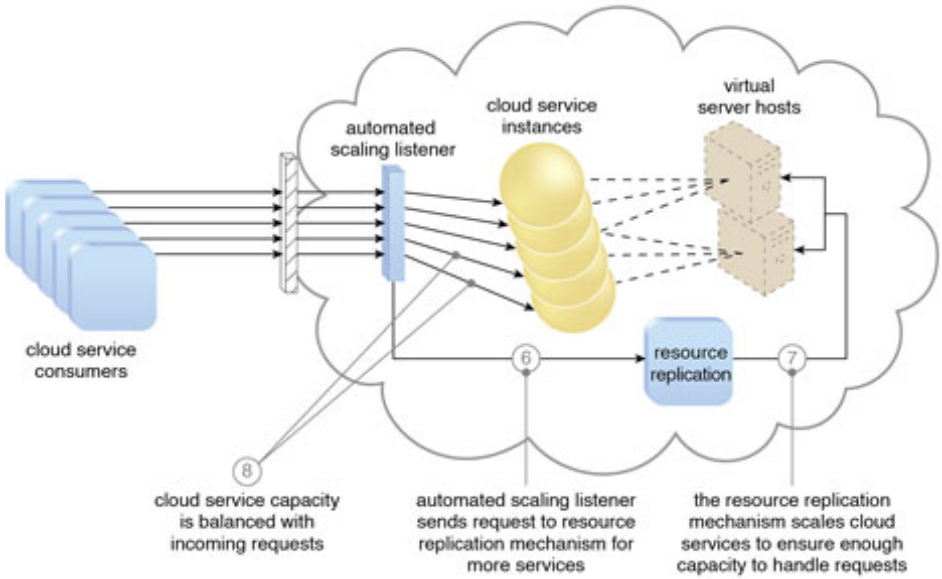



**Ilustración 10 - Diagrama concepto Cloud *cloudpatterns.org**

Este cambio de concepto, a dividir la ejecución de un sistema en varias máquinas, ha hecho que algunas aplicaciones opten por ejecutar tareas costosas en el lado del servidor para así no depender de la potencia del dispositivo cliente y poder mantener los tiempos de las operaciones constantes independientemente del terminal.

Sin embargo, ha hecho que las aplicaciones tengan que evolucionar y que las tradicionales bases de datos relacionales se queden obsoletas, debido a que no están pensadas para el escalado horizontal.

#### Bases de datos NoSQL

Por el motivo mencionado anteriormente, surge un interés muy fuerte por estudiar y utilizar los sistemas de gestión de bases de datos NoSQL: bases de datos no-relacionales que relajan [lita2010] (o no garantizan) el principio ACID [prid2008] característico de las RDBMS y que hace que sean perfectas para funcionar en entornos distribuidos como la nube. Sobre todo, en entornos de lecturas y escrituras masivas o tiempo real.

Según Adam Lith en [lita2010], el término NoSQL fue usado por primera vez en 1998 cuando Carlo Strozzi lo utilizó para su base de datos open source que no ofrecía interfaz SQL. Más tarde, fue reintroducido por Eric Evans en un evento que discutía sobre las bases de datos open source distribuidas, en 2009.

A lo largo del trabajo se hablará más de NoSQL. Sobre todo, en los apartados de Arquitectura y Tecnologías, pero a modo introductorio estas son sus características principales según Guoxi Wang en [wang2012]:

- Bases de datos sin esquema
- Escalabilidad horizontal
- Tolerancia a la partición
- No están preparadas para ACID
- Consistencia eventual
- No están pensadas para JOINS

## 3.5. Conclusiones respecto al Cloud Computing y SGBDs

Las sensaciones que deja la actualidad se dejan ver a lo largo del trabajo. De forma resumida, las tendencias del mercado son a ajustar presupuestos y, por tanto, el concepto de la nube estará muy presente.

El *cloud computing* está preparado para utilizar *commodity hardware*, esto es: hardware convencional y barato; y escalar de forma horizontal añadiendo más máquinas cuando estas sean necesarias. Debido a que la migración cuesta dinero, poco a poco, las empresas comenzarán a almacenar sus datos en la nube en vez de albergarlos en servidores dedicados propios.

De hecho, las administraciones públicas ya están trabajando en abrir al público general todos los datos y trabajos[46] que realizan con dinero público para que los desarrolladores puedan trabajar con ellos y crear aplicaciones ofreciendo servicios en relación a los esos datos. A esta iniciativa, también se le conoce como Open Data.

El movimiento open source, ha sido el principal responsable de la evolución del software, la comunidad de la que se nutre es muy grande y avanza siempre hacia las necesidades reales de los consumidores.

Por tanto, pese a que se augura un proceso lento y costoso, la competencia que está emergiendo con las plataformas open-source de las compañías Mozilla Foundation y Canonical, augura una tendencia a ir ganando terreno gracias al uso sistemas abiertos como las aplicaciones web.

[46] Open Data: http://opendata.euskadi.net/w79-opendata/es/contenidos/informacion/que_es_opendata/es_que_es/que_es_opendata.html (apr 29, 2013)

# 4. Definición de la prueba de concepto

Con el fin de reforzar la investigación que se realiza en este proyecto fin de máster, se llevará a cabo la implementación de una prueba de concepto cuyo objetivo será poner a prueba y adquirir conocimientos de las tecnologías y arquitectura escogidas para la solución de un entorno de aplicación tipo.

El contenido de este apartado describe la prueba de concepto que se va a realizar. Comienza presentando el tipo de aplicación al que se quiere dar solución y poner a prueba y, prosigue introduciendo las características de la aplicación concreta que se implementará.

## 4.1. Tipo de aplicación a tener en cuenta

Tomando como base el desarrollo de una aplicación específica para móviles, se pretende implementar una solución cliente-servidor cuyo cliente consuma los servicios del servidor y el servidor nutra una base de datos con contenidos de fuentes de datos externas.

De la inmensa cantidad de tipos de aplicaciones existentes en la red, este caso de prueba se centrará en el estudio de aquellas aplicaciones cuyo fuerte es el consumo intensivo de datos preparadas para escalar horizontalmente por picos de actividad puntuales en el sistema.

Los datos almacenados, debido a que la posibilidad de que éstos varíen en contenido, han de ser capaces de tener estructuras diferentes. Por tanto, pese a que los elementos conservarán una estructura similar, el esquema de los datos debe ser flexible y estar preparado para cambiar.

Para llegar al mayor número de personas, la aplicación cliente deberá ser multiplataforma y poder ser ejecutada en cualquier dispositivo móvil. Para lograr este requisito, se centrarán los esfuerzos en desarrollar una aplicación web debido a que puede ser ejecutada en cualquier navegador. Para ello será necesario que los

dispositivos cuenten con un navegador compatible con la tecnología HTML5 sobre la que se centrará el trabajo.

El servidor, por su parte, para poder atender a las restricciones que se plantean, hará uso de un sistema de gestión de bases de datos orientado a las NoSQL y se investigará un lenguaje de servidor innovador y veloz como puede ser Node.js del que se hablará más adelante.

Para poner el servidor accesible a internet, es imprescindible que esté alojado en un servidor que pueda escalar horizontalmente, por lo que se deberá realizar la implantación en un sistema que esté basado en la nube. Debido a que en el pasado ya se ha trabajado con este tipo de servicios, por experiencia con el sistema de Amazon Web Services, se albergará la aplicación en este servicio de alojamiento en la nube.

## 4.2. Introducción a la aplicación a implementar

Una vez definido el marco sobre el que se trabajará la prueba de concepto, se describen las características principales de la aplicación que se desarrollará.

La aplicación se especializa en la búsqueda y presentación de unos resultados que irán ligados a la geo-posición del dispositivo. En este caso, se realizarán búsquedas en relación a establecimientos cercanos al terminal cliente. Estos resultados se podrán filtrar en función de tipos de establecimientos. Así, si el parámetro de búsqueda solicita restaurantes, el servidor devolverá aquellos restaurantes que estén situados geográficamente más cerca de la coordenada objetivo.

El cliente listará los resultados según cercanía y permitirá ampliar la información de un establecimiento, mostrando su localización en un mapa navegable que podrá mostrarse en modo pantalla completa.

Además, para mejorar la experiencia del usuario, se le permitirá almacenar localmente bajo demanda los resultados para poder ser consultados posteriormente.

La base de datos del servidor, por su parte, se nutrirá de los datos ofrecidos por diferentes fuentes de datos. Se contemplarán varias alternativas de suministro como APIs de empresas terceras que ofrezcan datos geolocalizados o fuentes abiertas de información como Linked Open Data.

Además, como se describía anteriormente, será importante que el SGBD sea capaz de realizar búsquedas en función de una coordenada, devolviendo así los resultados ordenados por distancia de una forma sencilla y lógica, evitando cálculos pesados que ralenticen su funcionamiento.

# 5. Tecnologías

El desarrollo de una aplicación cliente servidor cuenta con numerosas posibilidades y alternativas. Sin embargo, se descartarán las tecnologías tradicionales para estudiar las nuevas alternativas y las tecnologías utilizadas en los desarrollos más innovadores.

En este apartado se describirán las posibilidades que se tendrán en cuenta para el desarrollo del caso de prueba mencionado anteriormente. Se desarrollará un subconjunto del alcance de una aplicación que probará las tecnologías que a continuación se presentan, comenzando con la parte del servidor, denominada como *backend*, y posteriormente con la del cliente, *frontend*.

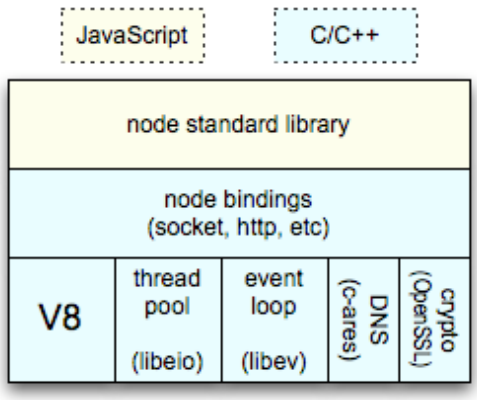
## 5.1. Tecnologías en el Backend

Para el desarrollo del lado del servidor se escogerán dos tecnologías: el lenguaje con el que se llevará su desarrollo y un sistema de almacenamiento para guardar los datos de los que el frontend consumirá.

### Lenguaje base: Node.js

Existen numerosos lenguajes con los que poder llevar a cabo este desarrollo del servidor pero, por el carácter innovador que pretende tener esta aplicación, se descarta el desarrollo de la solución con lenguajes típicos como Java, C# u otros ampliamente conocidos.

Por tanto, la solución de la aplicación del servidor se implementará con Node.js, un entorno que surge en 2009 y desde entonces ha llamado la atención de la comunidad de desarrolladores pese a su inmadurez por ser algo diferente a los lenguajes conocidos y por su acercamiento hacia las aplicaciones de red y su escalabilidad [Ojaa2012].




La arquitectura de la plataforma Node.js es destacada por su modelo de ejecución basada en eventos. Utiliza un único thread (el event

*thread*) para la ejecución del código. Esto quiere decir que no hay concurrencia a nivel de aplicación y, por tanto, todas las solicitudes son tratadas por el mismo hilo.

Sin embargo, este funcionamiento de un único hilo haría que Node.js no fuera eficiente y sufriera continuos bloqueos por no poder realizar solicitudes concurrentemente cuando se estén realizando cálculos de mucho volumen o bloqueos debido a E/S. Para tratar este tipo de tareas, Node.js utiliza un patrón en el cual se añade un parámetro de *callback* a las funciones y el motor de Node.js ejecuta la función en segundo plano y llama al callback cuando su ejecución concluye [ojaa2012].

El lenguaje nativo de Node.js es JavaScript, fue implementado en C++ y está basado en el motor V8 JavaScript de Google, lo que le hace muy veloz y eficiente.

#### SGBD del servidor

En la aplicación que se pretende desarrollar, es necesario que el servidor pueda almacenar datos de forma masiva donde los ítems tomen diferentes estructuras y datos dependiendo del origen de los datos, tipo de ítem y la cantidad de información que se pueda obtener de él.

Actualmente, dos tipos de sistemas de gestión de bases de datos (SGBD) copan las posibilidades de almacenamiento para servidores: los sistemas relacionales y los no-relacionales. Sin embargo, por ser considerado motivo de investigación y por lo comentado anteriormente respecto a los tipos de datos a recoger y que a día de hoy ocupan un alto porcentaje de las investigaciones tecnológicas, se acotará la decisión del sistema de almacenamiento a los SGBD no-relacionales.

##### *Principales características de los SGBD NoSQL*

El concepto NoSQL surge en 2009, rompe con el modelo relacional y rebaja las características de los principios ACID [wang2012].

Los principios fundamentales de NoSQL son el teorema CAP, el teorema BASE y la consistencia eventual.

El teorema CAP, introducido en el año 2000 por Eric Brewer [bree2000], está formado por tres conceptos: Consistencia, Disponibilidad y tolerancia a la Partición; principalmente, se basa en que un sistema distribuido no puede cumplir las tres características mencionadas.

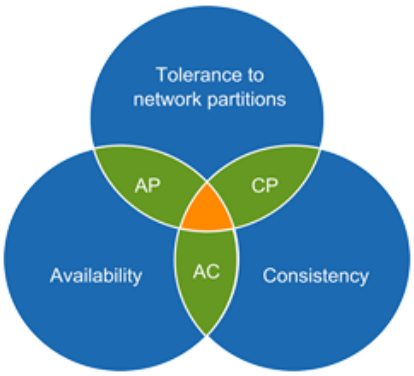


**Ilustración 11 - Teorema CAP**

Por tanto, de acuerdo con éste teorema, se pueden formar tres tipos de pares de diseño para nuestra aplicación: AP, CP o AC. Debido a la importancia de la disponibilidad en las aplicaciones web, descartamos CP y, puesto que el desarrollo estará basado en la nube, daremos importancia a la tolerancia en la partición: AP.

Las siglas del teorema BASE, por su parte, hacen referencia a BasicallyAvailable, Soft-state y Eventually consistency. Soft-state representa que en un determinado momento, el estado del sistema podría ser asíncrono pero, al final, terminaría siendo consistente [prid2008].

La *consistencia eventual* es uno de los modelos conceptuales que se utiliza en el dominio de la programación paralela que significa que en un período de tiempo largo, si no se ha enviado ninguna petición de cambio, se espera que todas las actualizaciones se propaguen eventualmente por el sistema, lo que hará que todas las réplicas sean consistentes.

Además, los SGBDs orientados al NoSQL están preparados para la escalabilidad horizontal y la ejecución en máquinas convencionales (commodity hardware) por lo que son ideales para funcionar en soluciones en la nube. De esta forma, deja de ser necesaria la contratación de servidores potentes capaces de ejecutar la aplicación y las migraciones a servidores más potentes cuando se quedan pequeños u obsoletos. El cloud computing permite tener servidores con hardware convencional e ir ampliando la red de servidores con más máquinas según la aplicación crece, bajo demanda. Éstos SGBDs, según Genqiang Gu en [gug2012], están preparados para este tipo de escalado y ésta es una de las ventajas de su existencia.

Un reciente estudio realizado por Alexandru Boicea [boia2012] para comparar el rendimiento de una base de datos relacional frente a una no relacional, demuestra que la ganancia en tiempo al contar con una gran cantidad de datos, tomando Oracle y MongoDB como representantes de un SGBD relacional y uno no-relacional respectivamente, es muy significativa.

### *SGBDs NoSQL considerados*

Se consideran a estudio dos sistemas de gestión de bases de datos no relacionales NoSQL: Cassandra (sistema basado en un modelo clave-valor) y MongoDB (orientado a bases de datos documentales) que son descritos a continuación.

#### Cassandra

Se trata de un SGBD altamente escalable, eventualmente consistente, distribuido, estructurado en un sistema de almacenamiento clave-valor, basado en la disponibilidad, tolerante a la partición y consistente [gug2012].

Según la documentación oficial de Cassandra[47] utiliza las tecnologías de sistemas distribuidos de Dynamo, el modelo de datos de BigTable de Google y no se queda solo en la estructura clave-valor sino que, además, ofrece la posibilidad de definir familias de columnas (ColumnFamily). No soporta modelos de datos relacionales pero, en cambio, incluye dos tecnologías que lo hacen realmente potente: partición de los datos, para mejorar en escalabilidad y rendimiento; y replicación de los datos, para obtener más disponibilidad y balanceado de carga.

[47] Cassandra: características: http://wiki.apache.org/cassandra/ (jun 19, 2013)

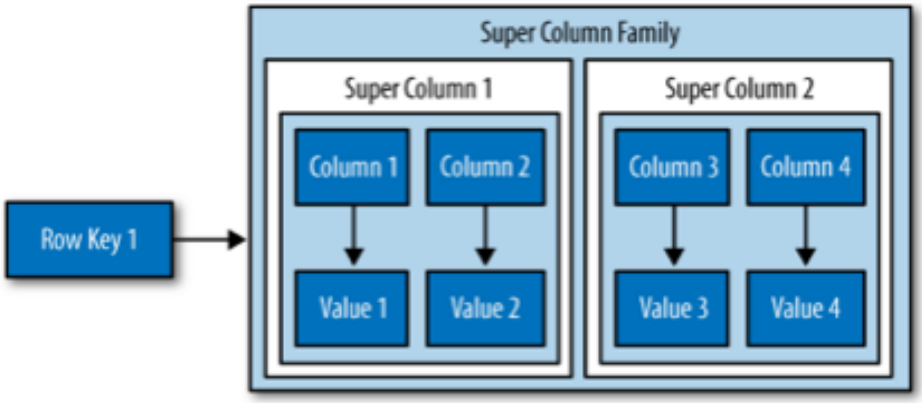



**Ilustración 12 - Modelo de datos en Cassandra *:**[48]

Las características principales que le diferencian de otros sistemas de gestión de bases de datos son las siguientes:

- Flexibilidad: Da la posibilidad de aumentar o reducir aleatoriamente los campos de cada dato.
- Escalabilidad
- Posibilidad de múltiples *data-centers*.

#### MongoDB

Liderado por 10gen, Inc., es un proyecto open-source de bases de datos de orientación documental pensado para la escalabilidad y la agilidad en el desarrollo, ideal para el almacenamiento masivo e inserciones y operaciones en tiempo real [gug2012].

MongoDB almacena los datos en forma de objetos representando documentos cuyos esquemas son dinámicos y se almacenan en formato JSON. Es un sistema que pretende hacer de puente entre las bases de datos relacionales y las orientadas a clave-valor cogiendo los mejores aspectos de ambas visiones: las características de las BBDD relacionales como los *índices* y las *queries dinámicas* por una parte; y los *esquemas dinámicos* (agilidad) y la *escalabilidad horizontal* de las NoSQL, por la otra.

Pero además, Alexandru Boicea adhiere en [boia2012] que MongoDB "está orientado hacia la potencia, flexibilidad, velocidad y facilidad de uso; añadiendo

[48] Miramon Enpresa Digitala: NoSQL: La siguiente generación de bases de datos: http://www.slideshare.net/dipina/nosql-la-siguiente-generacin-de-base-de-datos (jul 6, 2013)

posibilidades de replicación de los datos en servidores con indexación y ofrece *drivers* para múltiples lenguajes de programación".

Añade también que es posible realizar *actualizaciones condicionales*, *claves compuestas*, soporta caracteres Unicode, búsquedas *full-text* e incorpora un sistema para utilizar funciones de *agregación*.

El principio de agregación que sigue MongoDB es BASE, introducido previamente, ofreciendo consistencia, durabilidad y *atomicidad condicional*. Permite almacenar datos de hasta 16MB, es escalable horizontalmente y soporta replicación y compartición de datos [boia2012].

## 5.2. Tecnologías en el Frontend

La decisión de las tecnologías a utilizar en el lado cliente va fuertemente ligada a la arquitectura elegida para el cliente. Tal y como se analiza en el apartado de 6. Arquitecturas del sistema, se llevará a cabo el desarrollo de una aplicación web por las ventajas mencionadas en dicho apartado.

### HTML5

En los últimos años, cada vez son más los desarrolladores que se decantan por el ya ampliamente conocido HTML5, plataforma que saldrá de fase experimental llegando a estándar en 2014[49] y que combinada con CSS3 y WebGL se convierte en un estándar muy potente.

Impulsada por la W3C y la WHATWG, HTML5 es una plataforma que aún no está estandarizada pero que las últimas versiones de los navegadores, más aún en los navegadores de los dispositivos móviles, ya implementan en un gran porcentaje de todas sus funcionalidades, entre las que, junto con CSS3 y WebGL, destacan las siguientes [andk2012].

- **Soporte nativo de vídeo**: Se ha creado un nuevo elemento vídeo. Su objetivo es permitir a los desarrolladores embeber un vídeo sin tener que recurrir a *plugins* de tercero como Apple Quicktime o Adobe Flash.

[49] W3C: Plan 2014 http://dev.w3.org/html5/decision-policy/html5-2014-plan.html (jun 24, 2013)

- **Almacenamiento local**: Similares a las cookies, HTML5 ofrece la posibilidad de almacenar información en los clientes para después recuperarla. Una vez la página ha cargado, se podrá acceder a la memoria con JavaScript.
- **Capacidades offline**: En el caso de perder la conectividad, HTML5 permite al usuario continuar utilizando la aplicación web. Esto se logra gracias al manejo de la caché de HTML5, el desarrollador podrá especificar un *manifest* incluyendo una lista de ficheros necesarios para que funcione sin conexión. El navegador guardará copias de dichos ficheros.
- **Soporte de geolocalización**: No es estrictamente de HTML5 pero cuenta con una API de geolocalización que habilita a la aplicación web para preguntar al navegador por la localización actual del terminal.
- **Nuevos tipos de inputs**: HTML5 define numerosos nuevos tipos de inputs como cajas de búsqueda, cajas de carga, *sliders*, selectores de color, números de teléfono, direcciones de email o *datepickers* con diferentes formatos.
- **Web Sockets**: Ofrecen la capacidad para crear canales de comunicación multiplexados, bidireccionales y *fullduplex* a través de una única conexión TCP.

Conviene remarcar que, son los navegadores los que han de implementar estas funcionalidades de HTML5 para que puedan ser utilizadas por los desarrolladores. Pese a que está previsto que el estándar HTML5 esté cerrado en 2014, aún queda mucho trabajo por hacer a los navegadores para lograr implementar todas las funcionalidades.

Tras realizar una serie de búsquedas se ha comprobado que las compañías de los navegadores llevan sus desarrollos en privado y no existen planificaciones donde los desarrolladores puedan consultar para saber cuándo va a estar una determinada funcionalidad implementada.

Según [juna2013], "la predicción es que incremente de 336 millones de terminales con HTML5 en 2011 a 1.000 millones de ventas en 2013 y se estima que en 2016 alcance los 2.1 billones de dispositivos con navegadores compatibles con HTML5".

#### Almacenamiento en local

Existen tres especificaciones sobre HTML5 para almacenar datos en bases de datos locales del propio navegador: localStorage, IndexedDB y Web SQL Database. Según las especificaciones que se adjuntan en cada uno de los apartados que se introducen a continuación, las bases de datos que se crean en cualquiera de las tres soluciones implementan un mínimo de seguridad y son accesibles únicamente desde el origen del documento de dominio de cada una de ellas. Esto garantiza que una determinada aplicación web no podrá acceder a los datos de otra.

#### *localStorage*

Es una API que pertenece a la especificación de Web Storage que define un almacenamiento de datos persistente de tipo clave-valor para clientes web. localStorage[50] es la API más sencilla de HTML5 para el almacenamiento local, muy similar al funcionamiento de las *cookies*, sin tiempo de expiración y está implementado en la mayoría de las versiones de los navegadores actuales. Por ejemplo, en el caso de Internet Explorer, a partir de la versión IE8.

A diferencia de las cookies, Web Storage es más seguro. Los datos no se incluyen en cada conexión con el servidor, únicamente cuando se solicitan. Un sitio web sólo puede acceder a los datos almacenados por él mismo.

La especificación de este servicio no impone ningún tipo de restricción concreta a los navegadores, sin embargo, se les aconseja que limiten la cantidad de espacio total disponible para cada sitio para evitar el uso abusivo y que, al llegar al límite de espacio, se pregunte al usuario si desea ampliar dicha *quota*.

[50] HTML5 localStorage: http://dev.w3.org/html5/webstorage/ (2 Jun, 2013)

***IndexedDB***

Es la solución para una base de datos basada en NoSQL. Almacena datos en forma de clave-valor e indexa las claves para mejorar el rendimiento del acceso a los datos. IndexedDB[51] implementa un árbol de estructura B-tree persistente.
Además, permite crear índices adicionales para acceder a los datos a través de atributos diferentes a la clave.

A diferencia de localStorage, indexedDB ordena los documentos según su clave, es eficiente a la hora de buscar valores y permite la inserción de documentos con la misma clave. Por tanto, se trata de una especificación enfocada a la gestión de datos avanzada.

Esta solución ofrece la posibilidad de crear numerosas bases de datos identificadas por un nombre y el propio documento de dominio cuyos datos, identificados por una clave, pueden ser objetos semejantes a los utilizados en MongoDB.

IndexedDB ha de permitir que los valores de los datos que se almacenen en la base de datos sean de cualquier tipo simple como Strings primitivos, Numbers, Date o instancias de Arrays u Objetos. Además, se pueden definir diferentes tipos de índices para los atributos de un elemento.

Como la mayoría de los lenguajes NoSQL, IndexedDB, no dispone de lenguaje SQL.

***Web SQL Database***

Se trata de una especificación de almacenamiento de datos semejante a cualquier SGBD de tipo SQL. Era mantenida por la WHATWG. Sin embargo, tal y como dice el artículo de esta especificación[52], en Noviembre de 2010 W3C paraliza el proyecto Web SQL Database centrándose en las dos soluciones presentadas anteriormente debido al poco apoyo recibido y por la existencia de un sistema de almacenamiento de datos SQL masivamente utilizado como SQLite.

[51] HTML5 IndexedDB: http://www.w3.org/TR/IndexedDB/ (2 jun, 2013)
[52] HTML5 webDatabase: http://www.w3.org/TR/webdatabase/ (3 jun, 2013)

Pese a que la especificación está paralizada y no se prevé retomarla, algunos navegadores como Chrome, Safari y Opera la incluyeron en su implementación de HTML5. Sin embargo, no se le da mantenimiento, está estancada y navegadores como IE y Firefox no la soportan.

# 6. Arquitectura del sistema

Para poder describir las diversas arquitecturas que existen y dar solución al desarrollo a realizar como prueba de concepto, se cree conveniente comenzar con una descripción de la aplicación para situar al lector en contexto y describir la arquitectura de la aplicación.

Con el fin de probar las tecnologías que se estudian, se implementará un subconjunto de las características y funcionalidades que se describen a continuación.

## 6.1. Descripción de la aplicación

La aplicación móvil a desarrollar pretende dar solución y soporte a personas que desean encontrar establecimientos de interés cercanos a su localización. Para ello, basado en un sistema de búsquedas que utilizará diferentes fuentes de datos, el consumidor podrá obtener información sobre un sitio concreto a buscar, bien buscar cualquier tipo de establecimiento cercano o filtrar la búsqueda en función del tipo de establecimiento.

Si bien la aplicación buscará establecimientos en función de la geo-posición del terminal, ésta funcionalidad podrá ser desactivada para configurar manualmente una localización concreta.

Se dará la opción de almacenar establecimientos como favoritos, como visitados o almacenar listas o rutas (colecciones ordenables) de establecimientos en la memoria local del cliente. También se podrán marcar establecimientos como "pendientes" con el fin de visitarlos algún día.

Se ofrece la posibilidad de darse de alta en la aplicación para poder ofrecer funcionalidades extra. El registro podrá ser independiente o bien estar vinculado a una cuenta de Facebook aprovechando su API y dejando a la propia red social gestionar los asuntos de privacidad de la persona. También existirá la posibilidad de vincular la cuenta a una cuenta de Twitter o de Google. Este registro hará

posible que, además de almacenar en la memoria local, tal y como se describía anteriormente, dichos datos se puedan compartir en las redes sociales, realizar una compartición pública que se mostrará en una sección concreta de la aplicación donde se visualizarán todos los lugares que han sido recomendado por los usuarios registrados, o bien recomendar a un contacto concreto de la lista de contactos.

Para esta última opción, será necesario disponer de una funcionalidad de contactos en la que se agregarán amistades. La aplicación contará con un lugar en el que se podrán realizar búsquedas de contactos a través de las cuentas vinculadas para solicitar su amistad.

Además las colecciones que se creen podrán ser compartidas públicamente. De esta forma un usuario podrá realizar una búsqueda de listas o recorridos de establecimientos cercanos a su localización. Estas listas podrán ser tanto comentadas por los usuarios como valoradas y, a su vez, podrán ser recomendadas vía redes sociales por terceros.

## 6.2. Arquitecturas identificadas

Para llevar a buen puerto la aplicación detallada anteriormente existen diferentes arquitecturas que podrían hacer viable su desarrollo. La prueba de concepto es una aplicación cliente-servidor por lo que en este apartado, se estudiarán diferentes alternativas para cada lado de la aplicación.

Además, se buscarán diferentes servicios de donde poder obtener información sobre los lugares o establecimientos cercanos. Ya sea bases de datos de establecimientos concretas, bases de datos Linked Open Data, o APIs de terceros como pueden ser: Google Places[53], Foursquare[54] o Facebook Places[55].

### Lado del servidor

A continuación se van a describir las tres alternativas que se han identificado para la arquitectura del servidor y que han sido denominadas: doble respuesta, servidor puente y respuesta única.

#### *Alternativa: Doble respuesta del servidor*

El servidor funcionaría haciendo la labor de almacén de datos donde se guardarían los resultados de absolutamente todas las solicitudes que le llegaran a lo largo de su utilización. De esta forma, se dispondrá de información que podrá ser usada para devolver una primera aproximación al cliente y, después ser completada con una segunda respuesta con los nuevos resultados obtenidos del resto de fuentes de datos.

#### *Alternativa: Servidor puente*

El servidor se encargaría de intermediar entre las APIs desde las cuales se consumirían los datos y los clientes que solicitan dicha información. De esta forma no se almacenaría ningún dato en memoria, solicitaría información a las fuentes de datos, se normalizaría el contenido y se devolvería al cliente.

[53] Google Places API: https://developers.google.com/maps/documentation/javascript/places?hl=en#TextSearchRequests (May, 2013)
[54] Foursquare Search Venue: https://developer.foursquare.com/docs/venues/search (May, 2013)
[55] Facebook Search API: https://developers.facebook.com/docs/reference/api/search/ (May, 2013)

#### *Alternativa: Respuesta única*

Sería una mezcla entre las dos alternativas anteriores. El servidor sólo devolvería una respuesta con los datos obtenidos de las fuentes de datos y los almacenados en su propia memoria.

### Lado del cliente

El cliente, a su vez, también cuenta con diferentes arquitecturas posibles que consumirían los servicios del servidor y que se explican a continuación. En todas las alternativas el lado cliente tendrá la posibilidad de almacenar determinados datos en una base de datos local con el fin de poder utilizar la aplicación offline y dar más rapidez en la interacción con la misma.

#### *Alternativa: Nativa*

La primera alternativa es la tradicional, se realizarán varios desarrollos en paralelo de forma nativa para cada plataforma / sistema operativo móvil (iOS, Android, Windows Phone…), para lograr una multi-plataformidad y que la aplicación pueda ser utilizada por cualquier usuario desde cualquier terminal.

#### *Alternativa: Aplicación web*

Pretende desarrollar una única aplicación, aplicación web por tanto, que será compatible con cualquier terminal.

En este apartado existen otras dos alternativas pero, además, cabe la posibilidad de discutir dónde se albergará la aplicación:

- Existirá un servidor independiente para la misma desde el que se consumirán los datos del servidor de datos.
- Se compartirá el servidor de datos con el de la aplicación haciendo que el acceso a datos sea más veloz.

##### Alternativa webapp pura

La aplicación web que se desarrolle estará disponible a través de una URL y se ejecutará en los navegadores de los terminales móviles no siendo, de esta forma, necesaria ninguna instalación en el terminal.

##### Alternativa *web-clipping*

De la misma forma que la alternativa anterior, se habilitará una URL para la aplicación pero no se hará pública. En su lugar, se desarrollarían aplicaciones

nativas para cada sistema operativo. Estas aplicaciones simplemente cargarían un *webview* apuntando a la URL habilitada. También conocidas como web-clipping.

***Alternativa: Webapp - nativo***

Se desarrollará una única aplicación pero, en este caso, se utilizaría una de las plataformas existentes en la red como podría ser Phone Gap para traducir esta aplicación a código nativo de diferentes plataformas.

## 6.3. Arquitectura definitiva

Una vez analizadas las diferentes alternativas de arquitecturas posibles para el desarrollo de la aplicación propuesta, es necesario optar por la que se va a usar en la prueba de concepto.

Se cree que la solución ideal para el backend de la aplicación planteada es la de *doble respuesta*, sin embargo, es la solución más compleja y ya que la alternativa de *única respuesta* es compatible con ella, se implementará esta idea y en desarrollos futuros se tratará de resolver la ideal.

Respecto al lado cliente, el frontend, debido a las características de este estudio, es evidente que la solución a escoger será una que se base en web. Por tanto, se descarta realizar la *alternativa nativa* y, para lograr compatibilidad absoluta, también se descarta la alternativa *webapp-nativo*.

Por experiencia propia, el *webclipping* suele ser uno de los principales motivos para el rechazo de las aplicaciones en la plataforma de Apple, por lo que la alternativa definitiva, será la *webapp pura*. Se desarrollará una aplicación web y se accederá desde ella a través del navegador. Sin embargo, se utilizarán técnicas que simulen que la aplicación web es nativa.

## 6.4. Diagrama de la arquitectura definitiva

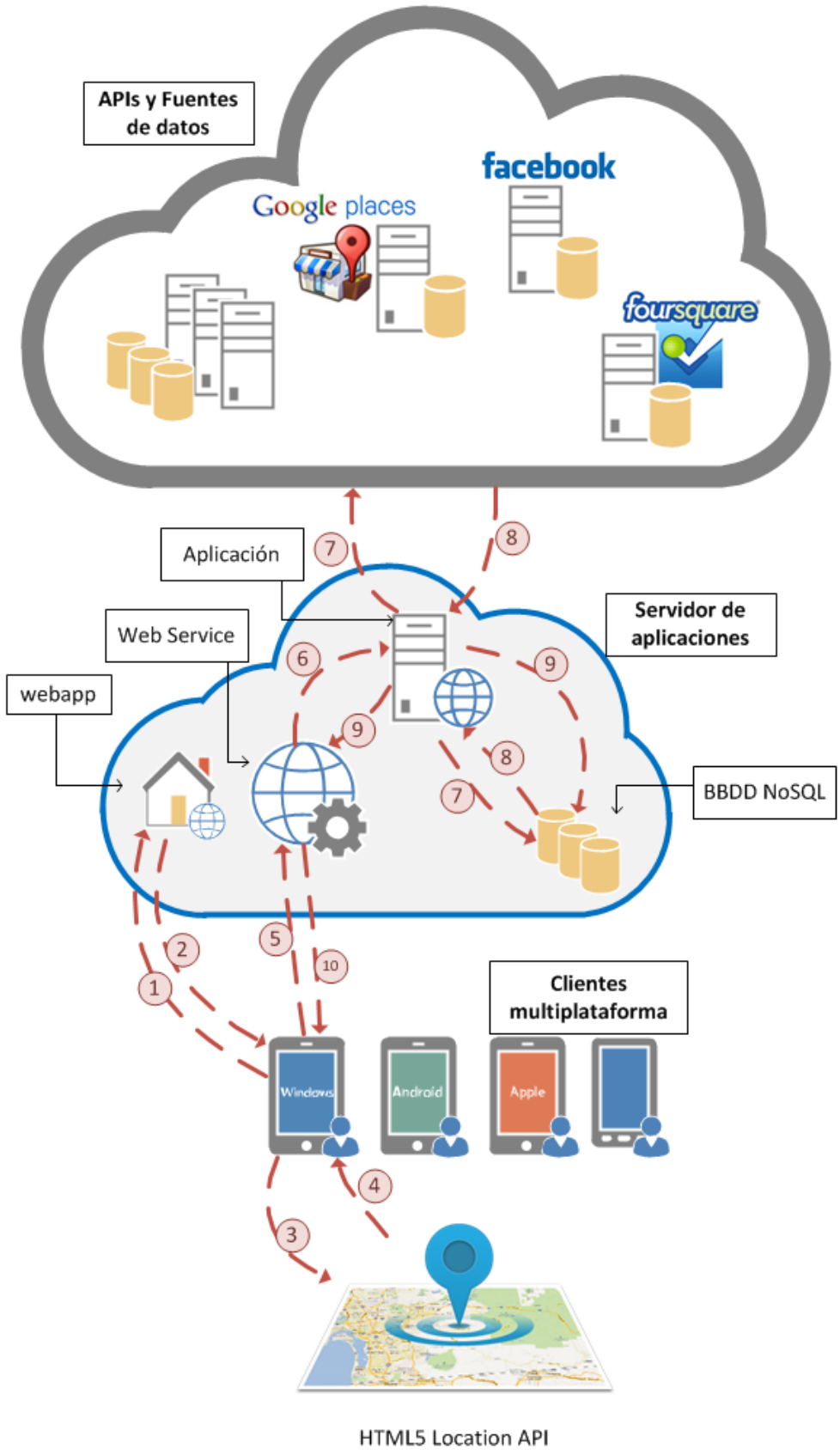



Ilustración 13 - Arquitectura de la prueba de concepto

# 7. Servidor

El servidor se montará sobre una máquina Linux Ubuntu Server 12.04 y, tal y como se mencionaba en el apartado 4. de tecnologías, se implementará sobre Node.js. Además, con el fin de lograr un servidor más potente y flexible, se hará uso de varios módulos de Node.js que se añaden en forma de capas sobre él.

En este apartado se detallarán los módulos que se utilizarán para desarrollar el caso práctico de las tecnologías a investigar, se describirá el diseño del servidor, cómo se ha desarrollado y qué se ha utilizado además de mostrar ejemplos, se detallarán las pruebas de concepto realizadas y finalizaré con unas conclusiones sobre la experiencia adquirida.

## 7.1. Frameworks sobre Node.js

Node.js es el núcleo sobre el que se implementa el servidor y para enriquecer el sistema, dependiendo de la aplicación a implementar, es conveniente instalar sobre él una serie de frameworks que se describen en este apartado.

### Express.js

Se va a utilizar el framework Express[56] que proporciona una fina capa de características fundamentales para la creación de aplicaciones web.

Una vez instalado el framework, comenzar un proyecto nuevo es tan sencillo como crear una carpeta con un fichero .json de configuración, con una serie de parámetros que describen la aplicación, y ejecutar un comando de instalación. Esto generará toda la estructura del proyecto facilitando la tarea para utilizar un patrón Modelo Vista Controlador.

[56] Express.js: http://expressjs.com/ (May, 2013)

El fichero de configuración tiene una estructura como la que se muestra en el ejemplo a continuación:

```
{
  "name": "hello-world",
  "description": "hello world test app",
  "version": "0.0.1",
  "private": true,
  "dependencies": {
    "express": "3.x"
  }
}
```

La librería está orientada al protocolo http, con lo que nos dará la posibilidad de atender a las solicitudes, devolver diferentes tipos de cabeceras y configurar URLs amigables haciendo uso de los verbos conocidos en el protocolo (GET, POST, PUT, DELETE, etc.).

### Jade – Node Template Engine

Jade Lang[57] es un potente motor de plantillas diseñado para ser utilizado con Node.js. Permite modular el código HTML en bloques para después reutilizarlos, crear variables y mostrarlas, simplifica la generación de código e incluso, permite crear condicionales o bucles al leer variables.

En Jade es muy importante la tabulación para la generación del código, gracias a ello el motor será capaz de identificar la estructura del documento y poder compilarlo a HTML ya que no requiere cerrar las etiquetas. Jade convierte a HTML todo el código y lo devuelve al navegador cuando termina su ejecución.

En la siguiente figura se puede observar una comparación de un fichero codificado en HTML y otro en Jade.

**HTML**

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Jade</title>
    <script type="text/javascript">
      if (foo) {
        bar()
      }
    </script>
  </head>
  <body>
    <h1>Jade - node template engine</h1>
    <div id="container" class="col">
      <p>You are amazing</p>
      <p>Jade is a terse and simple
         templating language with a
         strong focus on performance
         and powerful features.</p>
    </div>
  </body>
</html>
```

**Jade**

```
doctype 5
html(lang="en")
  head
    title= pageTitle
    script(type='text/javascript')
      if (foo) {
         bar()
      }
  body
    h1 Jade - node template engine
    #container.col
      if youAreUsingJade
        p You are amazing
      else
        p Get on it!
      p.
        Jade is a terse and simple
        templating language with a
        strong focus on performance
        and powerful features.
```

Por la comodidad que supone y la velocidad que se adquiere al maquetar con Jade en vez de HTML puro, se utilizará Jade para diseñar las pantallas del cliente.

[57] Jade: http://jade-lang.com/ (May, 2013)

#### MongoDB

También existe un módulo que sirve de driver de MongoDB preparado para utilizar sobre Node.js del que se hará uso para gestionar los datos que la aplicación consumirá. Se creará un fichero que contenga exclusivamente la comunicación entre los datos y la lógica de la aplicación.

## 7.2. Diseño

Tal y como se refleja en la ilustración 14, el servidor estará dividido en tres capas. Además de las clásica capa de acceso a datos, que gestionará tanto la lectura como la escritura de los datos de una base de datos NoSQL MongoDB; y lógica de negocio, cuya misión será controlar todas las acciones que se realicen; se le añadirá una capa de enrutamiento que gestionará las peticiones del cliente. Esta última capa será la encargada de trasladar la solicitud a la capa de negocio y devolver la presentación correspondiente al cliente.

También se utilizará una clase-módulo que se encargará de la comunicación, conexión y consumo de los servicios web de las APIs que se utilizarán como fuentes de datos.

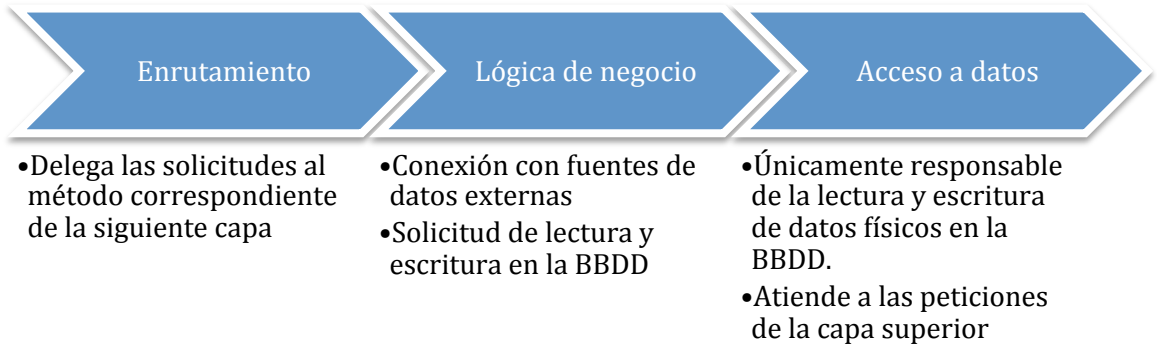



**Ilustración 14 - Diagrama de flujo de peticiones del servidor**

#### Consumir y almacenar datos

La clase de Conectores_API dispondrá de un método para cada fuente de datos a consumir y buscará establecimientos cercanos en cada una de ellas en función de los siguientes parámetros:

- Una *coordenada* en un *radio* en metros.
- Número *máximo de resultados* fijado inicialmente en veinte.
- Dos parámetros de filtrado de búsqueda, opcionales, en forma de String.
    - *Tipo* concreto de establecimiento.
    - *Query* de búsqueda.

***Flujo de eventos***

**Precondición**: Para poder hacer uso de esta clase y llamar a sus métodos, es imprescindible proporcionar una coordenada.

1. La capa de negocio recibe una solicitud de búsqueda en función de los siguientes parámetros, de los cuales sólo el primero es obligatorio: coordenada, query, tipo-establecimiento. A continuación, ejecutará el punto 2. y 4.
2. La capa de negocio instancia la capa de acceso a datos y solicita los datos que ya tenga en relación a los parámetros de búsqueda.
3. La capa de datos le devuelve los datos a la de negocio y ésta se los devuelve al que realizó la solicitud.
4. La capa de negocio instancia la clase de conectores API y llama a los métodos de cada una de las fuentes de datos para solicitar los resultados asociados a los parámetros.
5. Cada método de la clase de conectores crea una conexión HTTP con su correspondiente solicitud y le adjunta un *callback* que será llamado cuando se obtenga la respuesta de los servidores externos.
6. Una vez se recibe cada resultado, la clase se encargará de filtrarlos y crear una lista de objetos con los datos deseados listos para ser devuelta a la capa de negocio.
7. Una vez recibe los resultados la capa de negocio, ésta se encargará de instanciar la capa de acceso a datos solicitando que almacene los resultados.
8. La capa de acceso a datos se conectará a la base de datos, comparará si existe el establecimiento haciendo una búsqueda por coordenada y, si no existe, insertará la entrada.

**Postcondición**: La base de datos se poblará con los resultados obtenidos de la búsqueda externa, no existentes en la base de datos.

***Diagrama***

En el siguiente diagrama se puede apreciar una solicitud común al servidor para la obtención de establecimientos.

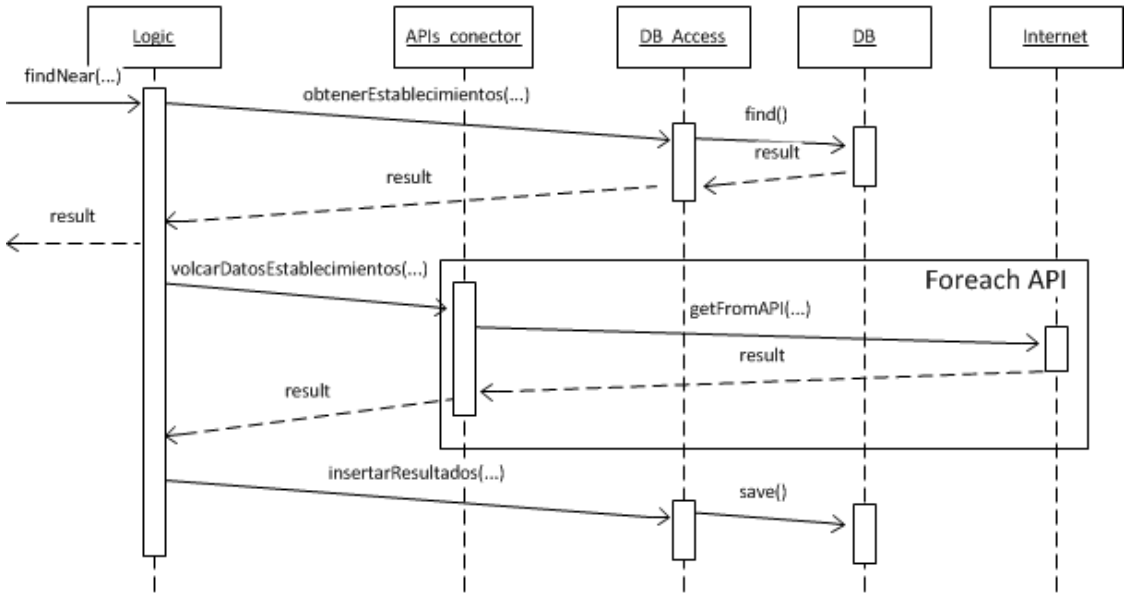



**Ilustración 15 - Diagrama de flujo de datos en el Servidor**

***Modelo de datos***

El servidor guardará los siguientes datos por cada establecimiento, en formato json y expresados a continuación en UML:

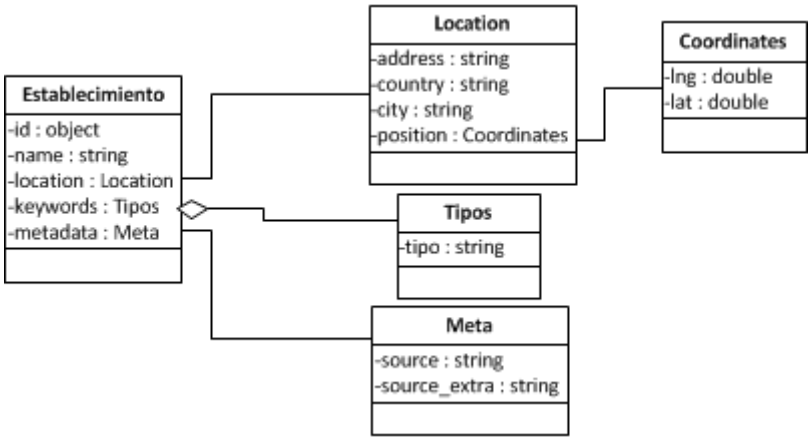



**Ilustración 16 - Modelo de datos básica del servidor**

Se definirá un índice de tipo 2d Index sobre el campo *location.position* con un índice secundario sobre *keywords* de tipo texto cuyo contenido se almacenará siempre en minúsculas. Se habla más sobre este tipo de índice en el siguiente apartado Desarrollo, Almacenamiento de los datos.

Para evitar establecimientos duplicados, MongoDB aconseja crear un índice de tipo único sobre el identificador del documento. Por este motivo se creará un índice de tipo único sobre el atributo *location.position* que comprobará la latitud y longitud. Se da por hecho que no existirán dos establecimientos sobre una misma coordenada.

Cada vez que se ponga en marcha el servidor se comprobará si existen los índices definidos en los últimos párrafos. De no existir, se crearán añadiéndoles el atributo "dropDumps"[58]: true cuyo cometido es borrar todos aquellos documentos que estén duplicados según su índice.

[58] Operadores en los índices de MongoDB:
http://docs.mongodb.org/manual/reference/method/db.collection.ensureIndex/ (May, 2013)

## 7.3. Desarrollo

Este apartado explica cómo se han implementado las tecnologías descritas previamente en el lado del servidor. Mostrará ejemplos, incluyendo código extraído de la prueba de concepto y simplificado para ilustrar de forma sencilla su cometido.

### Plataforma Node.js

Node.js es un lenguaje javascript de servidor y, como lenguaje javascript, utiliza un modelo de programación basado en eventos que hace que el sistema no se quede bloqueado en operaciones de entrada y salida, sino que continúe la ejecución de código dando una sensación de asincronía[59] [ojaa2012].

Como norma no escrita, los métodos y funciones implementadas en Node.js reciben como último parámetro una función a la que se le conoce como *callback*. Dicha función se ejecutará cuando el método finalice completamente su ejecución. El *callback* se utiliza para elevar tanto los errores como los resultados del método final y puede recibir el número de parámetros que se consideren oportunos.

En este desarrollo se mantendrá la costumbre de Javascript y Node.js conservándose la idea de la orientación a eventos. La política que se seguirá para la devolución de los resultados o elevación de errores será la de utilizar los callback, en cuya función se devolverán siempre dos variables: el valor del primer parámetro será el error encontrado, de haberlo, en otro caso se devolverá `null`; y el segundo parámetro devolverá los resultados de la ejecución del método.

Por tanto, a modo de ilustración, una función cualquiera tendría el siguiente aspecto:

```
logic.prototype.volcarDatosBusqueda = function( ll, query, type, callback ){
      apis.consumirFoursquare( ll, query, type, function( error, data ){
            if( error ) callback( error, null );
            else callback( null, data );
      });
}
```

[59] NodeJS: Qué es Node http://nodejs.org/about/ (jun 19, 2013)

El método que llame a `funcionEjemplo` tendrá que controlar también el resultado del callback para tratar o no el error:

```
main(){
      logic.volcarDatosBusqueda( "43.307,-1.973", "esparru", "bar",
                function( error, res ){
            if( error ) console.log( "El error devuelto es: " + error );
            else console.log( "El resultado es el siguiente: " + res );
   });
}
```

**Fuentes de datos**

Para nutrir de datos que el cliente pueda consumir, se han estudiado diferentes alternativas posibles. Entre ellas, se valora utilizar fuentes de datos Linked Open Data o APIs de aplicaciones existentes actualmente y en funcionamiento.

El concepto de Linked Open Data (LOD) fue mencionado por primera vez[60] por Tim Berners-Lee en 2006, planteando que la web semántica no trataba únicamente de poner los datos disponibles en internet sino de crear enlaces entre ellos y cumplir una serie de principios, como una URI única que identifique un objeto, que dicho objeto sea referenciable a través de una URI HTTP, utilizar el estándar RDF o SPARQL para presentar su información e incluir otros enlaces a otras URIs con el fin de descubrir más datos.

[60] W3C: Linked Data http://www.w3.org/DesignIssues/LinkedData.html (jul 6, 2013)

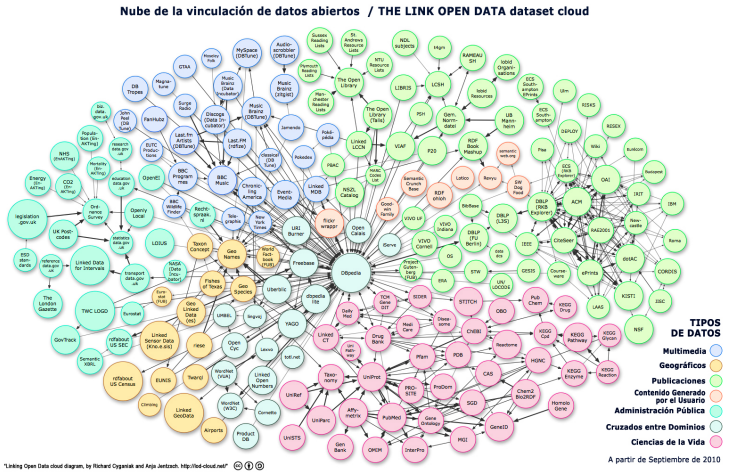



**Ilustración 17 - nube de Linked Open Data * lod-cloud.net**

En contraste con la web, Michael Hausenblas dice en [haum2010] que LOD pretende facilitar el acceso a la publicación y consumo de datos utilizando datos estructurados y URIs HTTP. Por tanto, este sistema está más enfocado a que sean utilizados por las aplicaciones que por las personas.

Sin embargo consumir LOD hace que sea necesario una búsqueda de fuentes, que los datos cumplan los requisitos especificados, mantenimiento y revisión constante de los datos para que éstos estén actualizados en la aplicación. Esta prueba de concepto no busca ese objetivo pero puede ser una clara línea futura a estudiar.

Por tanto, se toma la decisión de utilizar APIs de terceros, de empresas o aplicaciones expertas y con gran experiencia en el sector; que hacen uso y mantienen actualizadas, por ellos o por los propios usuarios, bases de datos orientadas a la localización. Tal y como se mencionaba en el apartado 5. Arquitecturas, se analizan tres APIs que ponen a disposición del desarrollador sus datos para ser consumidas: Google Places, Foursquare y Facebook.

Para llevar a cabo este desarrollo, se ha decidido crear una librería independiente que contenga los métodos necesarios para comunicarse y gestionar los datos que proporcionan las APIs. Este fichero hará uso de los módulos *http*[61] y *https*[62] de Node.js para crear conexiones con las respectivas APIs.

[61] Node http: http://nodejs.org/api/http.html (May, 2013)
[62] Node https: http://nodejs.org/api/https.html (May, 2013)

***Google Places***

La API de Google Places[63] permite realizar búsquedas de establecimientos sujetos a una localización formada por una coordenada (latitud, longitud) y un radio que limita los resultados por cercanía en metros.

Además, permite otros parámetros opcionales de búsqueda, como palabras clave, idioma de los resultados, nombre o tipo. También permite ordenar los resultados según importancia o cercanía.

Para tener acceso a la API de Google Places es necesario estar, en primer lugar, registrado en Google y, en segundo lugar, tener una clave de API que se obtendrá creando un proyecto[64] en Google Code. Se trata de un servicio que funciona sobre HTTPS REST al que se accede vía GET y permite obtener los resultados tanto en json como en xml. Devuelve una documento de tipo application/json con una lista de lugares que cumplen los parámetros de búsqueda y un *status* de respuesta (OK si es correcta).

***Foursquare***

La API de Foursquare para realizar búsquedas en función de una coordenada se conoce como Search Venues[65] y es la que se utilizará para obtener datos de Foursquare. No soporta HTTP, sólo funciona sobre HTTPS y no permite acceso anónimo, por lo que es necesario estar registrado para poder solicitar un par de CLIENT SECRET y CLIENT ID que se utilizarán para la autenticación OAuth2, imprescindibles para consumir de su servicio web. Esta API está construido sobre una plataforma HTTPS REST, por lo que para consumir sus servicios se definirá una URL con los parámetros asociados al tipo de búsqueda a realizar y se hará una petición https de tipo GET. La respuesta, por su parte, es devuelta en forma de texto de tipo application/json que se parseará en el servidor con la función `JSON.parse()` de JavaScript.

[63] Google Places: https://developers.google.com/places/documentation/search?hl=es (May, 2013)
[64] Crear proyecto Google Places: https://code.google.com/apis/console/?pli=1 (May, 2013)
[65] Foursquare Search Venues: https://developer.foursquare.com/docs/venues/search (May, 2013)

Por motivos de compatibilidad con diferentes versiones de la API, Foursquare exige al desarrollador que añada un parámetro extra a las peticiones URL que incluya la fecha en la que se desarrolla el método que consume el servicio. Es decir, si se desea obtener los datos en el formato definido por la API en el día que se desarrolla el servicio, será necesario añadir un parámetro a la URL como sigue: &v=YYYYMMDD. Si no se incluye, se generará un aviso de que el método está *deprecated* y devolverá los datos en el formato de la primera versión de la API.

***Facebook***

Facebook dispone de una API con la que se puede interactuar para realizar acciones relacionadas con su plataforma. Su API es conocida como Facebook Graph API y en este trabajo se hará uso de la interfaz de búsqueda[66] del apartado de lectura de datos. De la misma forma que las anteriores analizadas, se trata de un servicio de tipo HTTPS REST que requiere autenticación. Para autenticarse es necesario tener una cuenta en Facebook y, a continuación, dar de alta la aplicación en la herramienta de desarrolladores.

Una vez se ha registrado la aplicación, se obtienen un par de CLIENT SECRET y CLIENT ID que deben convertirse en ACCESS TOKEN, consumiendo el método *access_token* del módulo *oauth*. Este procedimiento no viene explicado por parte de Facebook pero es imprescindible para hacer uso de la API de búsquedas.

Para lograr un access_token, se tendrá que consumir el siguiente servicio que nos devolverá inmediatamente su valor:

```
https://graph.facebook.com/oauth/access_token?grant_type=client_credentials&c
lient_id=<CLIENT SECRET>&client_secret=<CLIENT ID>
```

El método search de esta API permite buscar diferentes tipos de contenidos. Para la resolución de este trabajo se trabaja con el tipo de contenidos “place” que devuelve *establecimientos*. Además, permite filtrar las búsquedas en función de una coordenada y un parámetro de búsqueda.

[66] Facebook Search Graph API: https://developers.facebook.com/docs/reference/api/search/ (May, 2013)

***Conclusiones de las fuentes de datos utilizadas***

Las tres fuentes de datos de las que se obtiene información de establecimientos necesitan una forma de autenticar la aplicación, por tanto, son necesarios sendos registros en cada plataforma.

Como requisito cumplido, las tres APIs investigadas ofrecen la posibilidad de realizar una búsqueda de establecimientos en función de una coordenada y acotar las búsquedas a un máximo de resultados o a resultados que cumplan un radio de cercanía. Además, permiten al menos introducir un parámetro de filtrado que se utilizará para buscar por tipos de establecimientos o nombres de los mismos.

Por último, también coinciden en que utilizan REST sobre el protocolo HTTPS seguro y que las respuestas devuelven la posición exacta en forma de coordenada de cada establecimiento.

Como conclusión, quizá sea la API de Facebook la que ofrezca resultados más pobres, ofreciendo menos cantidad de resultados y con menos metadatos sobre cada establecimiento. Por el contrario, Foursquare es capaz de devolver más información sobre cada establecimiento, además de datos relacionados con la propia aplicación (que se ignorarán). Sin embargo, Google permite hacer una búsqueda exhaustiva a posteriori sobre cada establecimiento utilizando otro método de su API.

**Almacenamiento de los datos**

El almacenamiento de los datos se realizará sobre una base de datos NoSQL MongoDB. Cabe mencionar que la aplicación y la mayoría de sus funcionalidades irán relacionadas con la geo-posición del cliente. Es decir, será necesario almacenar información de la localización de los lugares que se importen.

Antes de continuar, es importante mencionar que todo documento en MongoDB está identificado unívocamente por un atributo _id de tipo `ObjectId`[67] cuya característica por defecto es que es un BSON[68] de 12 bytes que, además, puede ser utilizado para conocer la fecha de creación e inserción de un documento.

En este caso, a parte de obtener datos en formato de texto como son la dirección, ciudad, país, etc. en la medida de lo posible se guardará una coordenada que haga referencia a la posición exacta de un establecimiento.

MongoDB, debido a la fuerte demanda de búsquedas geospaciales, a partir de la versión 2.2 implementa un tipo de índice para almacenar puntos en un plano de dos dimensiones cuyo uso está especializado en las búsquedas geo-localizadas. Este tipo de índice es denominado 2d Index[69] y se utiliza en bases de datos con información de localización que no los almacenen como objetos GeoJSON[70].

Como se puede leer en la documentación, el índice 2d da soporte a cálculos en superficies planas, plano Euclidiano, y devuelve los resultados ordenados por distancia en forma ascendente con respecto a la coordenada de búsqueda. Además, al definir un índice 2d, se permite definir un segundo índice sobre otro campo adicional sobre el que se hará un filtro a posteriori.

MongoDB permite un único índice 2d por colección, pero es posible añadir índices adicionales de otros tipos. El índice adicional del índice 2d ha de ser un único elemento, imposibilitando que éste sea una lista o array de strings. Por este motivo, debido a que interesa filtrar los resultados según un parámetro coincidente con el elemento de tipo array "keywords", se decide no añadir el índice secundario y se creará otro índice independiente para el atributo *keywords*.

Node.js dispone de un driver para trabajar con MongoDB. No viene instalado por defecto así que será necesario descargar el módulo para utilizarlo en el proyecto.

[67] MongoDB: ObjectId http://docs.mongodb.org/manual/reference/object-id/ (jun 24, 2013)
[68] MongoDB: BSON http://es.docs.mongodb.org/meta-driver/latest/legacy/bson/ (jul 9, 2013)
[69] 2d Indexes de MongoDB: http://docs.mongodb.org/manual/core/2d/ (may 28, 2013)
[70] GeoJSON: Especificación http://geojson.org/geojson-spec.html (jul 9, 2013)

La última versión más actualizada del driver de MongoDB para Node.js (May 2013) es la 1.3.6, por lo que se ha hecho uso de ella.

***Uso y operadores de MongoDB***

MongoDB no cuenta con el lenguaje SQL, sin embargo, la forma de realizar consultas a una colección de datos es muy intuitiva.

Una vez se ha creado la conexión y se ha abierto una colección de MongoDB, los operadores principales[71] que se utilizarán, sobre una colección, son: `find`, `findOne`, `save` y `remove`. La aplicación también gestiona índices y, para la creación y eliminación de los mismos, se hará uso de los métodos `ensureIndex` y `dropIndex`. Además, existe un método con el que es posible ver qué índices existen e información asociada a ellos: `indexInformation`.

De la misma forma que se describía en el apartado de Desarrollo de Node.js, los métodos de MongoDB siguen el mismo patrón de uso. Por tanto, el último argumento de los métodos mencionados se utilizará como *callback*, el error se devolverá siempre en primera posición y los resultados como segundo argumento de las funciones *callback*.

Respecto a las funciones citadas, los parámetros que reciben cada una de ellas son objetos Javascript con estructuras similares que se tratan para realizar unas u otras operaciones. En los próximos párrafos se va a obviar el uso del último parámetro como callback para simplificar las explicaciones.

**Método save**

El método `save`[72] es especial, debido a que es un *wrapper* de dos funciones: `insert` y `update`. Este método hace la labor de ambas dos dependiendo el objeto que se le pase. Sólo ha de pasársele un parámetro de tipo objeto. Dicho objeto puede contener los atributos que el desarrollador desee.

[71] Collection methods: http://docs.mongodb.org/manual/reference/method/js-collection/ (May, 2013)
[72] Método save MongoDB: http://docs.mongodb.org/manual/reference/method/db.collection.save/#db.collection.save (May, 2013)

Si el objeto que se le pasa no contiene el atributo `_id`, el método hará un `insert` y añadirá automáticamente el atributo `_id` al documento con su correspondiente contenido. En el caso de que dicho objeto sí contenga un atributo `_id` existen dos posibilidades: Si existe dicho `_id` en la base de datos, actualizará el documento con los nuevos o modificados atributos haciendo `update`, si no existe, hará un `insert` del nuevo documento. Un ejemplo de cómo utilizar este método se puede ver en la ilustración 18.

#### Método find

El método `find`[73] cuenta con dos parámetros opcionales. Si no se incluye ningún parámetro, el método devolverá todos los documentos de la colección sobre la que se ejecuta. En caso de incluir parámetros, el primero corresponde con la *query* deseada y el segundo a la *proyección*.

Este método devuelve un cursor a los documentos que coinciden con el criterio de búsqueda. En la ilustración 19 es posible ver una porción de código que ejecuta en forma de ejemplo el método find.

#### Método findOne

La función `findOne`[74] devuelve un único documento de entre los que coinciden con las condiciones definidas en el primer parámetro. En caso de que haya varios documentos que satisfagan la *query*, devolverá el primero insertado. El segundo argumento de esta función hace las veces de un objeto que define la proyección.

#### Método remove

Para método `remove`[75], a pesar de que la acción a realizar es totalmente diferente a `find`, la forma de uso es muy similar.

Este método elimina todos los documentos de una colección coincidentes con la query del primer parámetro. Por tanto, si se le pasa una query vacía, eliminará

[73] Método find MongoDB: http://docs.mongodb.org/manual/reference/method/db.collection.find/#db.collection.find (May, 2013)
[74] Método findOne MongoDB: http://docs.mongodb.org/manual/reference/method/db.collection.findOne/#db.collection.findOne (May, 2013)
[75] Método remove MongoDB: http://docs.mongodb.org/manual/reference/method/db.collection.remove/#db.collection.remove (May, 2013)

todo documento de la colección desde la que se utilice. Además, recibe un segundo parámetro opcional que, si está a `true`, eliminará un único documento coincidente con la query.

#### Tipos de índices en MongoDB

Respecto a los métodos para la creación de índices, se utiliza ensureIndex[76] para crear cualquier tipo de índice.

El primer parámetro recibe los atributos de los documentos de una colección que se desean indexar y el tipo de índice por cada atributo. Además de los tipos de índice existentes, también existen opciones adicionales que definen el comportamiento de los índices en su creación y que se incluyen como segundo parámetro en el método.

#### Ejemplos de uso

A modo ilustrativo se muestran unos ejemplos haciendo uso de los métodos de MongoDB presentados en este apartado.

El siguiente código ejecuta el método sabe que devolvería los documentos que se hayan insertado correctamente.

```
db.collection.save( {nombre: "Juan", apellido: "García"} , callback );
```

**Ilustración 18 - Ejemplo método save de MongoDB**

El ejemplo de la ilustración 19 ejecuta una búsqueda que lleva implícita un AND y devolvería todos los documentos completos (ya que no se ha especificado una proyección) de *collection* que contengan dos atributos nombre y apellido con sus valores correspondientes.

```
db.collection.find( { nombre: "Juan", apellido: "García"} , callback );
```

**Ilustración 19 - Ejemplo método find de MongoDB**

[76] Método ensureIndex MongoDB: http://docs.mongodb.org/manual/reference/method/db.collection.ensureIndex/#db.collection.ensureIndex (May, 2013)

En el caso de estudio de este trabajo, se crearán tres tipos[77] de índices:

- *Índice compuesto* sobre location.position.lat y location.position.lng con la opción de único.
- *Índice multiclave* sobre keywords, que es de tipo array.
- *Índice 2d* sobre location.position. Índice especial para búsquedas geolocalizadas.

Respecto a las opciones o acciones a realizar en la inserción de índices, éstos son las que se han utilizado:

**Unique: boolean** – define si el índice es único. De serlo, a la hora de insertar documentos, no permitirá insertar los que contengan un atributo único que ya exista en la base de datos.

**dropDups: boolean** – A la hora de crear el índice, eliminará los documentos duplicados según el índice definido dejando el primer documento con dicho índice.

**Background: boolean** – Define si a la hora de creación de un índice, el trabajo de indexación se hará en segundo plano. Esto es ideal para crear índices en colecciones que ya tienen muchos documentos y evitar dejar la aplicación congelada mientras se crean.

**Name: string** – Simplemente define el nombre del índice para poder tratar con él si fuera necesario.

[77] Tipos de índices en MongoDB: http://docs.mongodb.org/manual/core/indexes/ (May, 2013)

Ejemplo de cómo crear un índice en MongoDB:

```
db.collection.ensureIndex( { "location.position.lng" : 1 ,
                        "location.position.lat" : 1 },
                        { name : "indexLatLng",
                          unique : true ,
                          dropDups : true }
);
```

En este ejemplo se crea un índice combinado sobre los atributos `lng` y `lat` de tipo único. Además, recibe el nombre indexLatLng y, en caso de que hubiera documentos con índices coincidentes, los eliminaría dejando sólo el de menor _id.

***Búsquedas parciales en MongoDB***

Para hacer búsquedas sobre atributos de tipo texto en los que se desea realizar una búsqueda parcial o aproximada, no coincidente (equivalente al LIKE de SQL), MongoDB dispone del operador $regex [78] con el que, mediante *expresiones regulares*, se podrán realizar dichas búsquedas.

$regex, por defecto es *case sensitive*. Si se desea hacer una búsqueda *case insensitive* hace falta añadirle una opción $option: -i. Sin embargo, este uso penaliza la velocidad de las búsquedas por lo que es mejor evitar usarlo.

Por ejemplo, si se desea buscar textos que contengan la palabra "restaurant", bastará con hacer: `name: { $regex : '.*restaurant.*' }`.

***Búsquedas Geolocalizadas***

Debido a la fuerte demanda y las últimas tendencias de geolocalización de contenidos y media, MongoDB ofrece funcionalidades para poder ejecutar búsquedas geolocalizadas de documentos en sus colecciones.

Existen diferentes alternativas para localizar documentos en función de una coordenada. Se van a introducir las dos opciones que se han estudiado durante el trabajo, basadas en los operadores $near y geonear, respectivamente.

[78] MongoDB Regex: http://docs.mongodb.org/manual/reference/operator/regex/ (May, 2013)

#### Operador $near

En primer lugar, el propio método `find` incorpora una propiedad para realizar este tipo de búsquedas. La propiedad se llama $near[79] y basta con utilizarla sobre el índice 2d, definido añadiéndole la coordenada sobre la que buscar documentos cercanos. Se puede ver un ejemplo de su uso en la ilustración 20.

Además, esta propiedad permite hacer un segundo filtrado sobre otro atributo. Sin embargo, este atributo ha de ser de un único valor con lo que descarta realizar búsquedas sobre atributos multivalor como listas o arrays.

#### Operador geoNear

Como segunda alternativa, tenemos la opción geoNear[80]. A diferencia de $near, geoNear no se utiliza con el método *find* sino que hace uso de un método que hasta ahora no se ha mencionado en el trabajo.

Para utilizar este operador, tal y como se muestra en la ilustración 21, es necesario usar el método executeDbCommand del drive de Node.js para MongoDB. Este método no se ejecuta en función de una colección sino que la colección se le pasa en el parámetro del objeto de la query.

geoNear permite ejecutar una segunda query, sin ningún tipo de restricción a la propia query de filtrado de documentos en función de una coordenada.

[79] Operación $near: http://docs.mongodb.org/manual/reference/operator/near/#op._S_near (May, 2013)
[80] MongoDB geoNear: http://docs.mongodb.org/manual/reference/command/geoNear/ (May, 2013)

#### Ejemplos haciendo uso de $near y geoNear

En el siguiente ejemplo, que hace uso del método $near, se buscarían todos los documentos cercanos a la coordenada que entren dentro de un radio de un kilómetro de distancia. $maxDistance viene dado en *radianes*[81] y para convertirlo a kilómetros es necesario dividirlo entre 111.12.

```
db.collection.find( { "location.position" :
      { $near : [ query.coord.lng , query.coord.lat ] ,
        $maxDistance : (1/111.12) }
} );
```

**Ilustración 20 - Ejemplo operador $near de MongoDB**

Sin embargo, en el caso de usar geoNear, se ejecuta una consulta a la base de datos. La colección en la que se desea buscar viene definida en el atributo geoNear. Por último, la segunda query se define sobre el atributo query y busca los documentos cuyo array de *keywords* incluya la palabra "restaurant".

```
db.executeDbCommand( { geoNear : collectionName ,
      near : [ query.coord.lng , query.coord.lat ] ,
      maxDistance : (1/111.12) ,
      query : { keywords : ".*restaurant.*" }
} );
```

**Ilustración 21 - Ejemplo operador geoNear de MongoDB**

#### *Conceptos importantes*

Es importante destacar que MongoDB exige que el parámetro *longitud* esté por delante del de *latitud* en las búsquedas y que el índice 2d debe definirse en ese preciso orden.

También resulta importante dejar claro que a la hora de realizar una búsqueda en función de unas coordenadas, es necesario añadir también el filtro maxDistance que, en forma de radio desde la coordenada de búsqueda, devuelve aquellos

[81] Radián: Definición por Juan Carlos Gorostizaga
http://www.ehu.es/juancarlos.gorostizaga/apoyo/radian.htm (6 jun, 2013)

elementos más cercanos a ella que no estén más lejos que maxDistance. La unidad de dicha propiedad viene dada en radianes[82], por lo que hay que tenerlo en cuenta al asignarle un valor.

Con esto se logra filtrar aquellos resultados más lejanos a dicho radio y evitar que se devuelvan establecimientos significativamente lejanos. De no hacerlo, nos dará los más cercanos que existan en la base de datos, lo que puede implicar distancias de kilómetros.

## 7.4. Prueba de concepto

Antes de comenzar ningún tipo de desarrollo en el servidor, se han probado las APIs desde las que se van a consumir los datos. Para testear las APIs, se han realizado todos los procesos previos de registro en los sistemas y se ha utilizado un navegador convencional como Google Chrome para realizar las primeras peticiones HTTPS y comprobar que los datos que se obtenían eran correctos.

Para poder verificar los resultados, se comenzó introduciendo manualmente coordenadas cercanas a lugares que resulten familiares. Una vez que se logran los resultados deseados de las APIs, se procede a realizar pruebas consumiendo dichas peticiones HTTPS desde el servidor de desarrollo, una máquina virtual instalada en el ordenador desde el que se trabaja. Se comienzan a utilizar los métodos GET de Node.js en una aplicación de servidor básica para estudiar si funcionan los servicios. Para poder comprobar el funcionamiento, se imprimen los resultados en la salida por defecto de texto del propio terminal.

Cuando se obtienen los primeros resultados con la primera API, se prueba a modificar los parámetros y a compararlos con los resultados que se obtienen desde el navegador.

Una vez que se logra poner en marcha las APIs, se comienzan a tratar los resultados, modificando los objetos y transformándolos y mostrando sólo los datos que se desean almacenar de forma local.

[82] $maxDistance unidad, radianes:
http://docs.mongodb.org/manual/reference/command/geoNear/ (May, 2013)

Cuando se ha logrado mostrar los objetos con los contenidos deseados, se comienzan a probar a realizar inserciones en MongoDB, primero manualmente, consultando posteriormente que los datos se han almacenado correctamente y, después, automáticamente con cada petición que se realice desde la aplicación cliente.

## 7.5. Conclusiones

Las fuentes de datos consultadas han resultado ser flexibles y funcionales para el caso de prueba de concepto planteado. Sin embargo, pese a haber descartado la solución de Linked Open Data, ésta podría ser la línea de trabajo a seguir en futuros estudios: Nutrir la aplicación con datos abiertos y propuestos por entidades abiertas, iniciativas sociales y fuentes de datos públicas.

Por tanto, como línea futura y trabajos posteriores, se tratará de sustituir o combinar las fuentes de datos utilizadas para este trabajo con fuentes *linked open* data. Esta decisión implicará un trabajo de población de datos manual y, por supuesto, un mantenimiento periódico para poder ofrecer y conservar unos datos actualizados.

Respecto a las tecnologías utilizadas para el desarrollo de la aplicación del servidor, se ha comprobado la flexibilidad y potencial de MongoDB para el tipo de aplicaciones planteada. Además, ofrece la posibilidad de almacenar datos geolocalizados, y varias alternativas y métodos para después consultarlos. Se han analizado los operadores geoNear y near, pero existen otras opciones, además de otras formas de almacenar datos localizados. En suma, se trata de una herramienta muy potente.

MongoDB cuenta con una página oficial con una documentación muy completa e incontables ejemplos. Además, para ayudar en la adaptación a este SGBD, muestran ejemplos de cómo se traduciría todo tipo de queries SQL a MongoDB.

Node.js, por su parte, también se contempla como una seria alternativa a los tradicionales lenguajes de servidor. Desde luego, es una herramienta muy sencilla y en poco tiempo es posible montar un servidor funcional. La comunidad de desarrolladores con la que cuenta es amplia, su página oficial, al igual que con MongoDB, ofrece una documentación completa, y existen numerosos módulos y drivers que añaden a Node.js más flexibilidad, funcionalidades y características.

# 8. Cliente

El cliente se albergará en la misma máquina que el servidor. Los usuarios accederán a la aplicación desde una URL y se trabajará para que esta misma dirección pueda funcionar *offline* después de realizar la primera carga, por tanto, será necesario *cachear*[83] la mayor parte del contenido de la aplicación, además de hacer uso del almacenamiento de datos local que pone HTML5 a disposición del desarrollador y que fue presentado en el apartado 5. Tecnologías.

Tal y como se aclara en el apartado del Servidor, además de que el cliente se implementará en HTML5, se hará uso de un motor de plantillas de Node.js, Jade, una potente herramienta para facilitar la tarea de generación de las plantillas del cliente. Conjuntamente, se hará uso de una serie de librerías que facilitan el desarrollo del cliente y dotan a la solución con una calidad superior que la asemeja a una aplicación nativa.

En este apartado se describirán los métodos que se desarrollarán para el cliente, cómo se desarrollan y con qué funcionalidades. Por último se presentarán unas pruebas y conclusiones personales.

#### Librerías adicionales

Para complementar a HTML5, se hará uso de los framework Javascript jQuery y jQuery Mobile, a los que se hará una breve introducción antes de comenzar con los bloques de diseño y desarrollo del cliente.

#### *jQuery*

Se utilizará una librería para JavaScript de nombre jQuery[84]. Esta librería enriquece el lenguaje e incluye funcionalidades y atajos para facilitar y reforzar el desarrollo de aplicaciones web que usan JavaScript. En definitiva, simplifica la interacción con los elementos del documento HTML y la programación en JavaScript.

[83] El término "cachear", del inglés cache, se utilizará para hacer referencia al almacenamiento local de ficheros y recursos en el terminal cliente.
[84] Sitio oficial de jQuery: http://jquery.com/ (jun 1, 2013)

Tal y como se especifica en su propio sitio oficial, esta librería es multi-plataforma, está preparada para funcionar en IE, Firefox, Safari, Opera y Chrome, entre otros. Es ligera y rápida y hace que los documentos HTML sean manipulables, maneja eventos, animacio,nes y hace que las peticiones con la tecnología asíncrona AJAX (Asynchronous JavaScript And XML) sean mucho más flexibles.

Se utilizará la versión 1.9.1 de esta librería cuya fecha de lanzamiento fue el 18 de febrero de 2013.

***jQuery Mobile***

Se trata de un framework multi-plataforma que extiende de jQuery y está orientado a aplicaciones de dispositivos móviles, tablets y táctiles. jQuery Mobile[85] está basada en HTML5 y usa sus capacidades, además de impulsar la web semántica con las etiquetas de HTML5.

La propia página responsable de jQuery Mobile declara que las siguientes compañías e iniciativas utilizan dicho *framework* en el desarrollo de sus aplicaciones versión móvil: *Ikea*, *Dinsey World*, *Slideshare*, *Box.net*, *Standford*, *American Century Investments*, *Rugby World Cup*, entre otras. Además, es un proyecto impulsado por corporaciones como *filament group*, *BlackBerry*, *Nokia*, *Mozilla Corporation*, *palm* o *Adobe*.

La aplicación utilizará la versión 1.3.1 de jQuery Mobile que fue lanzada el 10 de abril de 2013.

[85] Sitio oficial de jQuery Mobile: http://jquerymobile.com/ (1 jun, 2013)

## 8.1. Diseño

Una vez que el servidor es capaz de funcionar por sí solo, el cometido del cliente será consumir los propios servicios del servidor. Para ello, será necesario que el dispositivo en el que se va a ejecutar la aplicación disponga de un navegador compatible con HTML5 y las funcionalidades básicas de almacenamiento local y geo-localización.

Sin embargo, aunque el dispositivo no esté capacitado con la funcionalidad de GPS, la aplicación será capaz de funcionar. En este caso, será necesario definir en la configuración una localización manual (ya sea escribiendo una dirección física o proporcionando directamente una coordenada).

Además, se implementará la capacidad de que la aplicación pueda funcionar en modo sin conexión. La aplicación advertirá de que sin conexión no podrá realizar búsquedas, pero mostrará los resultados que el usuario haya decidido guardar localmente.

#### Estructura de la aplicación

En este subapartado se describe la estructura que tomará la aplicación cliente de la prueba de concepto que se refleja, esquemáticamente, en la ilustración 22.

La aplicación contará con una pantalla de configuración en la que se podrán definir formas de comportamiento de la aplicación.

- Habilitar la funcionalidad de GPS.
- Mostrar la posición del terminal en el mapa con el resto de resultados.
- Definir una posición manual en caso de apagar el GPS.

Al margen de los mencionados requisitos mínimos que ha de tener el dispositivo, la funcionalidad principal de la aplicación es la búsqueda de establecimientos cercanos. Para este cometido, existirán dos tipos de búsqueda:

1. Búsqueda de locales cercanos: no discriminará en resultados y devolverá los establecimientos más cercanos a la posición definida por el cliente (ya

sea de la propia geo-posición del terminal o una posición definida manualmente).

2. Búsqueda de locales cercanos filtrados por un parámetro de búsqueda y tipo de establecimiento: realizará una búsqueda de establecimientos próximos, filtrando los resultados según los parámetros de búsqueda definidos por el ejecutor de la aplicación.

Se le ofrecerá al cliente la posibilidad de guardar los resultados localmente, por si tiempo después desea recuperar la búsqueda exacta. Para ello, el cliente dispondrá de un apartado más en la aplicación en la que podrá consultar búsquedas guardadas anteriormente.

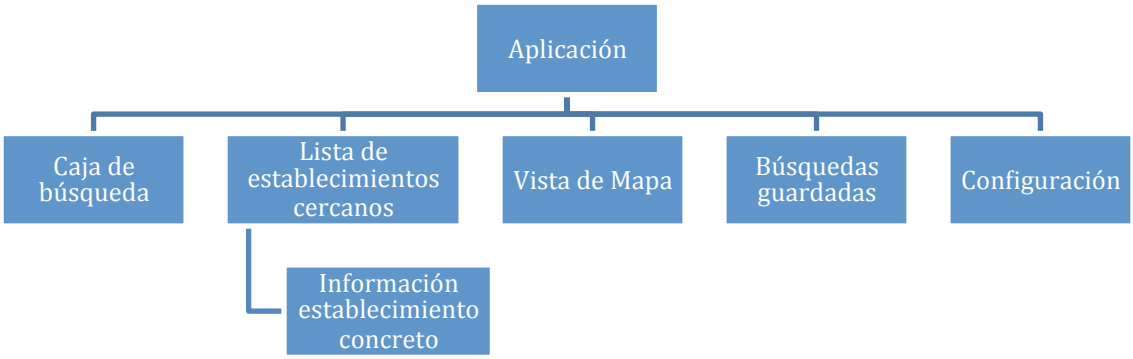



**Ilustración 22 - Esquema de la aplicación cliente**

Una vez mostrado el listado de establecimientos, además, se permitirá al cliente ampliar la información de un establecimiento pulsando sobre él. En esta pantalla se mostrará una vista miniatura de un mapa que posicionará el establecimiento, centrará la vista sobre el establecimiento y ofrecerá información adicional si se dispone de ella.

El mapa será dibujado gracias a la API de Google Maps API v3 y podrá ser ampliado a modo pantalla completa pulsando el botón de *zoom* con el fin de poder navegar sobre él y ver con más claridad dónde está situado.

Además, los resultados de listas de establecimientos permitirán transformar la lista, mostrada en formato de texto, a formato mapa navegable sobre el cual se añadirán todos los marcadores.

De forma resumida, obtenemos los siguientes casos de uso:

- Búsqueda global
- Búsqueda filtrada
- Mostrar un establecimiento concreto ampliado
- Vista en mapa
- Apartado de configuración
- Apartado de búsquedas archivadas

#### Almacenamiento de datos en local

Tras estudiar las tecnologías que HTML5 pone a disposición del desarrollador para el almacenamiento local, se toma la decisión de hacer uso de IndexedDB por su semejanza al SGBD utilizado en el lado del servidor y por ser un SGBD con funcionalidades avanzadas. localStorage presenta un sistema de almacenamiento simple y Web SQL Database, como ya se ha dicho, es un proyecto estancado y ha sido considerado *deprecated*, lo que implica que no se continuará y se eliminará de la especificación.

Debido a la necesidad de *cachear* determinados valores con el fin de recuperarlos en cada apertura de la aplicación, en lugar de almacenar variables en sesión o *cookies*, se hará uso de localStorage pos su sencillez y rapidez de uso. Se utilizará esta funcionalidad para guardar variables de configuración localmente.

No se contempla utilizar SQLite por ser una tecnología ampliamente conocida y carece de interés en este caso de estudio.

##### *Modelo de datos*

El clientes tendrá el siguiente modelo de datos para el almacenamiento local de listas de establecimientos:

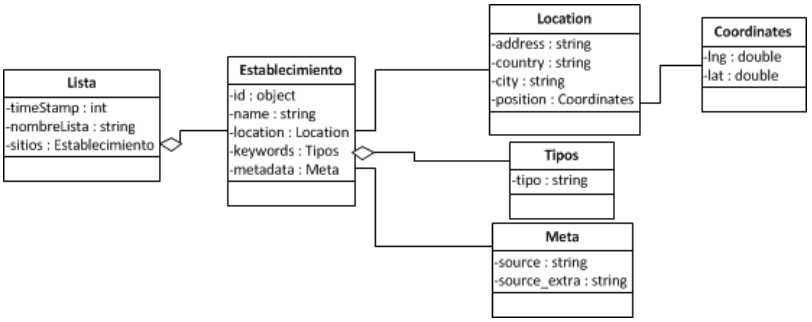



**Ilustración 23 - Modelo de datos del cliente**

Este modelo de datos refleja que se podrán crear listas de establecimientos cuya estructura de datos será la misma que la que mantienen los establecimientos en el lado del servidor.

## 8.2. Desarrollo

El cliente cuenta con un fichero javascript que hace las veces de controlador y su función será la de gestionar y preparar las interfaces y acciones cuando se solicite una actualización de la geo-posición del terminal.

El trasiego de información con la base de datos local del terminal se desarrollará en un fichero ad-hoc javascript específico para ésta tarea.

#### Obtención de la geo-posición del terminal

HTML5 ofrece una funcionalidad con la cual, es posible solicitar al navegador que proporcione la geo-posición del terminal. Si el navegador está actualizado y soporta dicha funcionalidad de HTML5, será posible realizar una búsqueda en función de la posición GPS.

#### *Geo-localización en HTML5*

La especificación de la interfaz de la API de Geolocalización[86] de HTML5 consta de tres métodos fundamentales para operar y tratar con la geo-posición del navegador: `getCurrentPosition`, `watchPosition` y `clearWatch`.

Tanto `getCurrentPosition` como `watchPosition` reciben como primer parámetro una función callback que será ejecutada cada vez que se complete la obtención de una posición y dejará la información de la geo-posición en un objeto de tipo `position` en el primer argumento de la función callback.

La diferencia entre ambos métodos radica en que el primero es una función a la que se le llama bajo demanda que, como resultado, devuelve una coordenada; y la segunda es una función que permanece a la escucha y se ejecuta cada vez que detecta un cambio de posición.

La función `watchPosition`, además, devuelve un identificador de tipo `long` que identifica al observador y que se utilizará en caso de querer dejar de observar. Para ello se ejecutaría la función `clearWatch`, cuyo parámetro será el identificador del observador.

[86] HTML5 Geolocation API: http://dev.w3.org/geo/api/spec-source.html (May, 2013)

***Implementación de la funcionalidad de geo-posición***

Lo primero que hará el controlador javascript del lado cliente al cargar la aplicación será comprobar si el navegador dispone de la librería `geolocation`. En tal caso, llamará a la función que devuelve la geo-posición en una función callback: `getCurrentPosition(callback)`.

Dicho método se ejecuta cada vez que se hace un cargado completo de la aplicación. Es decir, tal y como se aclara en la documentación[87] de JQM, debido a que jQuery Mobile no hace cargas completas de la página mientras se navega por la aplicación sino que obtiene los contenidos a través AJAX, el método sólo se ejecuta cuando se recarga o se abre la página, pero no mientras se navega por ella. Por tanto, durante la ejecución de la aplicación, y a medida que se cargan contenidos en el navegador, es necesario tratar los contenidos y adjuntar la coordenada a los enlaces y acciones de los formularios. Para este cometido se crea una función `updateInputLocation`, que actualizará los enlaces y formularios pertinentes en el callback de `getCurrentPosition` y en el evento que jQuery Mobile ejecuta cada vez que se termina de cargar un contenido y antes de mostrarlo `'pagebeforeshow'`[88].

***Adjuntar la coordenada en la búsqueda***

Al hacer uso del formulario de búsqueda de establecimientos, cuya acción se hará a través del método `GET`, se presentan dos opciones para adjuntar la coordenada del dispositivo:

- La primera opción que se considera es la de almacenar la coordenada en un input de tipo `hidden`. Sin embargo, al hacer `submit`, los navegadores, por motivos de seguridad, codifican[89] todos los inputs que se envían por `GET`. Existe la posibilidad de deshabilitar esta opción pero interesa que esté activa para el resto de campos y formularios.

[87] jQuery Mobile: Cargado de páginas http://jquerymobile.com/demos/1.2.0/docs/pages/page-navmodel.html (11 jun, 2013)
[88] jQuery Mobile: pagebeforeshow http://api.jquerymobile.com/pagebeforeshow/ (11 jun, 2013)
[89] W3C: Form encoding http://www.w3.org/TR/html401/interact/forms.html#h-17.13.3.3 (11 jun, 2013)

- Para evitar que la coordenada llegue codificada al servidor se toma la segunda alternativa: en lugar de guardar la coordenada en un input oculto, se modificará el atributo `action` del formulario añadiéndole el parámetro `ll` con la propia coordenada, haciendo que el resultado del `action` sea `search?ll=latitud,longitud`.

**Google Maps API v3**

La visualización de los establecimientos en mapa se realiza a través de la API de Google Maps v3[90], cuya última versión ha sido especialmente optimizada para ser utilizada en dispositivos móviles. Según la propia documentación de Google, para los sistemas operativos Android e iOS.

La API de Google Maps es muy configurable y flexible. Está escrita en javascript y permite definir en qué idioma se desean mostrar los controles, añadir marcadores, personalizar la interfaz con colores y controles *customizados* así como elegir entre distintos formatos de mapa (callejero, mapa).

Para la implementación, es necesario importar la librería al proyecto. Una vez importada, Google muestra diferentes ejemplos para poder realizar el primer trabajo con su API. Según los términos y condiciones[91] de Google, no está permitido ni cachear ni almacenar la librería en servidores externos.

En el caso del desarrollo que ocupa este trabajo, la implementación será más compleja que los ejemplos mostrados en la documentación, debido a que a medida que se cargan pantallas en el navegador, se mostrarán mapas en capas diferentes.

Por ello se creará una función para inicializar un mapa cuyos parámetros serán un identificador de capa (div) y una lista de coordenadas que harán las veces de marcadores en el mapa a visualizar.

[90] Google Maps API v3:
https://developers.google.com/maps/documentation/javascript/basics?hl=es (May, 2013)
[91] Google: términos y condiciones https://developers.google.com/maps/terms (11 jun, 2013)

Esta función, a la que se le denomina `initializeMap`, no tiene callback y su objetivo será el de cargar un mapa sobre la capa identificada por el primer parámetro.

Para que esta función funcione correctamente, es necesario que el DOM haya sido cargado completamente previamente. Para ello, la solución común es esperar el evento `ready` de jQuery. Sin embargo, como ya se había aclarado anteriormente, jQuery Mobile no recarga la página por completo sino que carga los contenidos vía AJAX lo que hace que no sirva esperar a tal evento. La solución es añadir un listener al evento de jQuery Mobile `pageshow`, que se ejecutará cada vez que finalice de cargar una página en el DOM y comprobará si dentro de ese nuevo contenido existe el ID de mapa y lo inicializará.

#### Almacenamiento local

Para almacenar datos localmente, es necesario disponer de ellos. Sin embargo, una vez se muestra el contenido en pantalla, los datos se pierden en el DOM. Para resolver esto, se presentan dos opciones: cada vez que se realiza una búsqueda, además de mostrar los resultados, almacenar en una variable los datos que se obtienen en el servidor; o habilitar un *web service* que únicamente descargaría los datos previa solicitud.

La primera solución cuenta con la ventaja de que el guardado en local se podría realizar inmediatamente después de realizar la acción correspondiente. Pero debido a que esta solución supone sobrecargar todas las comunicaciones cliente-servidor, por tener que adjuntar todos los datos en cada una de ellas, se estima más limpio y conveniente implementar la segunda.

Para esta solución, se requiere de un *web service* que devuelva la información del servidor en formato JSON, para poder tratarla y almacenarla en la base de datos local. Se habilita un parámetro `save=true` que se deberá adjuntar a la URL cuyos datos se desean archivar. De esta forma, solo con ese cambio en la URL, el servidor, en lugar de maquetar la solución, devolverá en formato JSON el resultado de la búsqueda.

El fichero de enrutador del servidor, antes de devolver el fichero near.jade con la maquetación preparada, comprobará si existe el parámetro `save=true` en la URL. Si se da el caso, devolverá los datos obtenidos en formato JSON. Una vez consumidos esos datos a través de una petición AJAX, se podrán tratar y almacenar en la base de datos local.

***Compatibilidad de IndexedDB***

Debido a que IndexedDB es una especificación relativamente nueva y que aún es un documento sin cerrar, no todos los navegadores la implementan. De hecho, en el estudio que lleva Alexis Deveria[92], Ingeniero de Software que trabaja para Adobe y ha realizado importante aportaciones a la W3C, en *canaiuse*[93], la mayoría de los navegadores en las versiones más altas a día de hoy la implementan. Sin embargo, dos de los navegadores más utilizados aún no han sacado una versión compatible del navegador con IndexedDB: Opera y Safari. Aunque está previsto que la nueva versión de Opera 15, lo incorpore.

| | IE | Firefox | Chrome | Safari | Opera | iOS Safari | Opera Mini | Android Browser | Blackberry Browser |
|---|---|---|---|---|---|---|---|---|---|
| | 8.0 | | | | | 4.2-4.3 | | 4.0 | |
| | 9.0 | 20.0 | 26.0 | 5.1 | | 5.0-5.1 | | 4.1 | 7.0 |
| Current | 10.0 | 21.0 | 27.0 | 6.0 | 12.1 | 6.0-6.1 | 5.0-7.0 | 4.2 | 10.0webkit |
| Near future | 11.0 | 22.0 | 28.0 | | 15.0 | | | | |
| Farther future | | 23.0 | 29.0 | | | | | | |

**Tabla 1 - Compatibilidad de IndexedDB en los navegadores * según caniuse.com a 6 de junio 2013**

Por su penetración en el mundo del smartphone al ser el navegador por defecto de iOS, resulta imprescindible que al menos Safari sea compatible con el almacenamiento local escogido. Sólo hay que realizar una sencilla búsqueda en internet para ver que la discusión al respecto es considerable, ya que la comunidad de desarrolladores demanda IndexedDB en Safari.

Es por este motivo que, en junio de 2012, un grupo de desarrolladores iniciaron un proyecto en GitHub[94] conocido con el nombre de IndexedDBShim[95], con la intención de crear una API intermedia que tradujera el código de IndexedDB para utilizar la especificación *deprecated* Web SQL Store.

[92] Alexis Deveria: http://www.linkedin.com/pub/alexis-deveria/12/340/483 (6 jun, 2013)
[93] Compatibilidad IndexedDB en Can I Use: http://caniuse.com/indexeddb (5 jun, 2013)
[94] Proyecto IndexedDBShim en GitHub: https://github.com/axemclion/IndexedDBShim/ (6 jun, 2013)
[95] IndexedDBShim con WebSQL: http://nparashuram.com/IndexedDBShim/index.html (5 jun, 2013)

Por tanto, se considera que el uso de IndexedDB en el contexto de la versión del cliente es prioritario, y se hará uso de ésta para poder utilizar la clase IndexedDB en Safari a la espera de que el navegador soporte la especificación elegida. Una vez todos los navegadores soporten IndexedDB la librería intermedia se dejará de utilizar.

***Introducción e implementación en IndexedDB***

Para utilizar la especificación de IndexedDB será necesario comenzar cargando los objetos necesarios para poder utilizar las instancias de la clase IndexedDB y abrir una conexión con la base de datos.

La forma con la que se inicializan estos objetos, es la siguiente:

```
window.indexedDB = window.indexedDB || window.webkitIndexedDB
                   || window.mozIndexedDB || window.msIndexedDB;
window.IDBTransaction = window.IDBTransaction || window.webkitIDBTransaction;
window.IDBKeyRange = window.IDBKeyRange || window.webkitIDBKeyRange;
```

En el código superior se observa que, para lograr mayor compatibilidad, se llama a los objetos con los nombres con los que están definidos en los diferentes navegadores. Esto es así ya que en la gama de navegadores para los que se implementa, el objeto IndexedDB viene definido con nombres de clase diferentes y, en cada uno de ellos, se concatenará cada objeto con objetos vacíos, obteniendo como resultado la clase completa.

Una vez se tienen las variables inicializadas, es necesario crear un constructor que llame al método `open` cuyo primer parámetro será el nombre de la base de datos que deseemos utilizar y el número de versión de la base de datos creada.

El número de versión servirá para actualizar la base de datos. Esto es, si IndexedDB detecta que dicho parámetro ha sido incrementado, ejecutará el evento `onupgradeneeded` en el que se podrán modificar las propiedades de la base de

datos: creación de nuevos `objectStores`, índices, o cualquier operación deseada. Un `objectStore` en IndexedDB es el equivalente a una tabla MySQL.

Todos los métodos de IndexedDB cuentan con dos eventos concretos que se ejecutan en diferentes momentos:

- **`onsuccess`**: Se ejecuta cuando una operación ha acabado con éxito.
- **`onerror`**: Cuando una operación ha detectado un problema o no ha finalizado correctamente.

#### Apertura de una base de datos

Así, tomamos como ejemplo la apertura de una base de datos:

```
localDB.indexedDB.open = function(callback) {
	var dbVersion = 1;
	var request = indexedDB.open('local_places', dbVersion);
	request.onupgradeneeded = function(e) {
		localDB.indexedDB.db = e.target.result;
		var db = localDB.indexedDB.db;
		if(!db.objectStoreNames.contains('lista')){
			db.createObjectStore('lista',
			    {keyPath: 'timeStamp', autoIncrement: false}
			);
		}
	};
	request.onsuccess = function(e) {
		callback();
	};
}
```

En el ejemplo superior, se puede observar como la versión que se ejecuta es la 1 y, la primera vez que ésta se ejecute, al detectar que no existe el `objectStore` "lista", lo creará. El evento `onupgradeneeded` se volverá a ejecutar en el momento que incrementamos su versión. Además, se define un índice sobre el `timeStamp` a través del cual accederemos a los datos.

La función `callback` se ejecutará en el momento en el que la operación de apertura finalice con éxito.

#### Escritura de datos

La escritura de datos en IndexedDB es muy similar a la introducida en la parte del servidor de MongoDB. A continuación se muestra un ejemplo, que se explicará posteriormente.

```
localDB.indexedDB.insertarLista = function(nombre, places) {
	var db = localDB.indexedDB.db;
	var trans = db.transaction(['lista'], 'readwrite');
	var store = trans.objectStore('lista');
	var request = store.put({
		'nombreLista': nombre,
		'timeStamp': new Date().getTime(),
		'sitios' : places
	});
	request.onsuccess = function(e) {
		console.log("Se ha añadido correctamente: " + nombre );
	};
	request.onerror = function(e) {
		console.log(e.value);
	};
};
```

Este ejemplo, dado un nombre y un array de sitios, descargados del servidor y tratados previamente para almacenar únicamente lo que nos interesa, reserva una transacción de tipo readwrite, selecciona el `objectStore` en el que realizar dicha transacción y escribe en él.

#### Lectura de datos

De la misma forma que en MongoDB, existen dos operadores para leer datos en IndexedDB.

- **`openCursor`**: Se utiliza para leer una colección de datos cuya query puede ser un rango de búsqueda. Por ejemplo, desde un índice hasta otro.
- **`get`**: Se utiliza para obtener datos que satisfagan una condición de tipo coincidente. Por ejemplo, los elementos iguales a un *timestamp*.

En el siguiente ejemplo, se muestra cómo obtener el contenido de una lista concreta.

```
localDB.indexedDB.obtenerListaConcreta = function(ts, callback) {
	var db = localDB.indexedDB.db;
	var trans = db.transaction(['lista']);
	var store = trans.objectStore('lista');

	var cursorRequest = store.get(parseInt(ts));

	cursorRequest.onsuccess = function(e) {
		renderListaConcreta(cursorRequest.result);
		callback();
	};
	cursorRequest.onerror = function(event) {
		console.log("error: " + event);
	};
};
```

Es importante que el parámetro que se introduce en la función `get` sea del mismo tipo que la clave. En caso contrario, no devolverá resultados. Por ello se hace uso de la función `parseInt` de javascript, que transforma un String en un Integer.

En este ejemplo, con el evento `success`, nos interesa dibujar en pantalla los datos obtenidos y, cuando esto acabe, ejecutar el callback. El sentido de este callback no es más que llamar a la función `refresh` de jQuery Mobile para dar estilo a la lista dibujada. Debido a que jQuery Mobile al dibujar la pantalla puebla las etiquetas con clases, no es necesario especificarlas al desarrollar ya que el propio framework las adjunta al inicializarse, pero no lo hace cuando la pantalla ya ha sido mostrada y, en este caso, la lista se puebla cuando la pantalla ya ha sido interpretada y mostrada en el dispositivo.

***localStorage para los datos aislados***

Por su simplicidad y facilidad de uso para almacenar datos de configuración de la aplicación o datos que interese cachear, como la geo-posición del terminal, también se ha hecho uso de la funcionalidad localStorage[96] de la clase Web Storage de HTML5.

Tal y como se introduce en el apartado de tecnologías, localStorage es un sistema de almacenamiento de tipo clave-valor y no admite claves repetidas. Por tanto, cuando se intenta introducir una clave que ya existe, ésta, en lugar de introducir un valor nuevo, modifica el existente. Se puede decir que la clave es de tipo única.

A continuación, se muestra un ejemplo sencillo de cómo realizar una escritura y lectura:

```
localStorage.setItem('usoGPS',usoGPS);
var gpsActivo = localStorage.getItem('usoGPS');
```

**Caché de los datos**

Para hacer que la aplicación *cachee* ficheros, HTML5 permite crear un fichero manifest de caché, conocido como Application Cache[97], en el que se define el procedimiento a llevar a cabo cuando se cargue el contenido web del sitio o aplicación.

Como se describe en la especificación de la W3C, este fichero ha de ser apuntado en un atributo `manifest` de la etiqueta `<html>` de cada página cargada. Se trata de un fichero de texto simple que dice al navegador qué ha de cargar, y qué no, y que contiene tres secciones, que se describen a continuación.

- **Cache Manifest**: Los ficheros que se listen bajo este título se cachearán una vez se hayan descargado por primera vez.

[96] HTML5 Web Storage – localStorage: http://www.w3.org/TR/2013/PR-webstorage-20130409/#the-localstorage-attribute (jun 8, 2013)
[97] HTML5 Application Cache: http://www.whatwg.org/specs/web-apps/current-work/#applicationcache (3 jun, 2013)

- **Network**: Los ficheros listados bajo este título nunca serán cacheados y necesitarán conexión a internet para cargarse.
- **Fallback**: Se listan pares de ficheros por línea. En caso de que el fichero de la izquierda no sea accesible, cargará el segundo. Por tanto, para que esto tenga sentido, el segundo fichero deberá aparecer en la lista de CACHE para que siempre sea accesible.

Es importante destacar que este fichero ha de comenzar con las palabras clave "CACHE MANIFEST" para que funcione. De comenzar con un comentario u otro contenido, no se interpretará y la caché no funcionará.

Tal y como se menciona en la WHATWG[98], el navegador comprueba byte por byte si el fichero manifest es el mismo que el obtenido del servidor. Si se da el caso de que el fichero ha sido variado, se volverá a interpretar el manifesto y se re-cachearán los contenidos nuevos a especificados.

Así, si el contenido fuera el siguiente:

```
CACHE MANIFEST
# 2013-06-01 v0.0.1
/stylesheets/style.css
/javascripts/ jquery-1.9.1.min.js
/offline.html

NETWORK:
/search

FALLBACK:
/ /offline.html
```

Los ficheros style.css, jquery y offline.html se cachean. En el caso de que uno de los ficheros que se añadan a la caché no existiera en el servidor, el navegador mostrará un error en la consola.

Se cree conveniente destacar que los ficheros que se añadan a la caché han de tener exactamente el mismo nombre con el que están escritos en el servidor. Por

[98] WHATWG: Actualización del manifest: http://www.whatwg.org/specs/web-apps/current-work/multipage/offline.html#downloading-or-updating-an-application-cache (9 jun ,2013)

ello, se ha de tener cuidado con los QUERY_STRING en las URL ya que si se intenta cachear un fichero al que se le adjunta una querie en la solicitud, éste no se cacheará ya que el nombre del fichero no coincide con el del *manifest*. Para evitar este comportamiento, una práctica común para adjuntar queries a las URL para ejecutar comportamientos diferentes según la querie, es utilizar el hashtag # en lugar del interrogante. Desde el navegador es posible acceder a los parámetros posteriores al hashtag con la propiedad `window.location.hash`.

Sin embargo, jQuery Mobile no es compatible con este uso debido a que el enrutamiento del framework utiliza precisamente el hashtag. La propia documentación oficial descarta su uso[99]. Esto hace que la única alternativa viable sea la de capturar el evento de click sobre un elemento, almacenar en localStorage los parámetros que se deseen y consultarlos una vez ha cargado la siguiente pantalla o enviarlos por POST al servidor.

El titular NETWORK implica que jamás se cachearán los ficheros que se listen bajo dicha palabra. En esta caso, no se guardará en caché /search ya que no tendría sentido realizar una búsqueda si no se dispone de conexión a internet; y en caso de querer consultar los lugares cercanos, se le redireccionaría a una sección donde se le mostraría un mensaje de que se está trabajando *offline*.

El titular FALLBACK ya ha sido explicado anteriormente. En el ejemplo, su cometido es el de cargar la pantalla de offline.html al acceder a la raíz de la aplicación. En offline.html, por ejemplo, se mostrará la página de inicio con una advertencia de problemas de conexión.

Una vez definido el manifest, como dice la documentación, el navegador ni siquiera solicitará al servidor los ficheros cacheados, los servirá directamente desde su propia caché aumentando la velocidad de carga, la prevención cuando no hay conexión a internet y el consumo de datos al ahorrarse la solicitud al servidor.

[99] jQuery Mobile: enrutamiento http://jquerymobile.com/demos/1.1.0-rc.1/docs/pages/page-navmodel.html (jun 12, 2013)

### Native Look Like

Pese a que una aplicación web no sea una aplicación nativa al uso, existen numerosas formas de hacer que ésta imite su comportamiento y se asemeje a la solución nativa. A continuación se describen varias formas de lograr que una aplicación web dé la sensación de ser nativa. Sumando todas las opciones se logran resultados muy semejantes a una aplicación nativa.

#### *Iconos e imagen de inicio*

Desde los inicios de los smartphones, Apple ha dado la posibilidad de añadir una página web al *home screen* del dispositivo. Para este cometido, los desarrolladores de iOS definieron una serie de `link-href` en los que se podía especificar un icono y un *splash screen*[100] para la aplicación web.

El splash screen es la imagen que se muestra nada más abrir la aplicación desde el home screen, mientras se descargan los contenidos durante la primera carga.

Debido a las diferentes resoluciones existentes en los dispositivos de Apple: con o sin pantalla Retina, iPhone o iPad; la propia etiqueta permite especificar el tamaño[101] de la imagen para su posterior correcta visualización. A continuación se muestra un ejemplo de cómo utilizar estas etiquetas que deberán de ser colocadas en la cabecera del código HTML.

```
<link rel="apple-touch-icon" href="touch-icon-iphone.png" />
<link rel="apple-touch-icon" sizes="114x114" href="touch-icon-iphone-
retina.png" />
<link rel="apple-touch-startup-image" href="/startup.png">
```

[100] Apple: Splash Screen e iconos en una aplicación web en iOS: http://developer.apple.com/library/ios/#documentation/AppleApplications/Reference/SafariWebContent/ConfiguringWebApplications/ConfiguringWebApplications.html (6 jun, 2013)
[101] Apple: Tamaños de los iconos http://developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/MobileHIG/IconsImages/IconsImages.html#//apple_ref/doc/uid/TP40006556-CH14 (6 jun, 2013)

***Viewport y zoom***

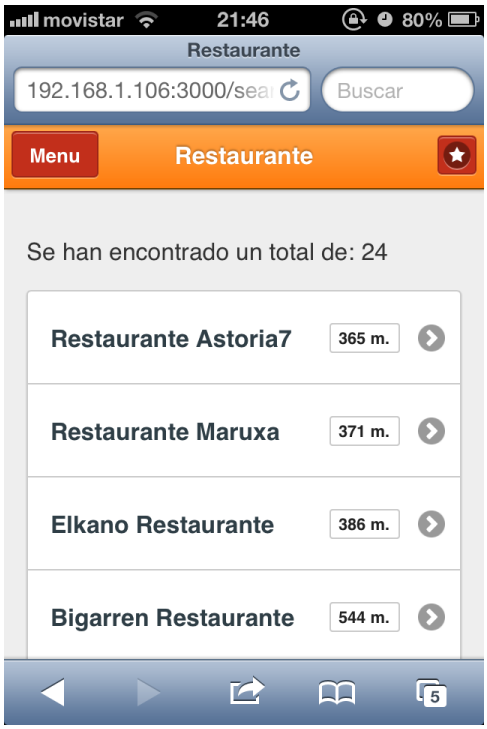
También es posible configurar determinadas formas de visualización de la aplicación. Así mismo, existe la posibilidad de bloquear el zoom, ocultar la barra de direcciones en determinados navegadores o definir el color que adquirirá la barra de estado de un dispositivo iOS.

Para este cometido Apple define una serie de `meta-tags` que, incluyéndose en la cabecera de la aplicación web, variarán la forma con la que se muestra la propia aplicación.

```
<meta name="apple-mobile-web-app-capable" content="yes" />
<meta name="apple-mobile-web-app-status-bar-style" content="black">
<meta name="viewport" content="width=device-width,
                      user-scalable=no,
                      initial-scale=1,
                      minimum-scale=1.0,
                      maximum-scale=1.0">
```

En este ejemplo la primera línea hace que en el navegador Safari de iOS se oculten los paneles de control, como la barra de direcciones y los botones de Safari; la segunda línea cambia el color de la barra de estado a negro cuando se ejecuta en modo pantalla completa; y la tercera deshabilita la posibilidad de hacer zoom y el escalado automático de cualquier navegador móvil.

A continuación, en las capturas de pantalla, la diferencia entre no incluir las dos primeras líneas y sí hacerlo.





***Detectar ausencia de conexión***

HTML5 dispone de un método al que se puede llamar para comprobar si hay conectividad en el dispositivo. Para acceder a éste método es suficiente con llamar a `navigator.onLine` cuando se estime oportuno, que devolverá true sólo si hay conexión.

Se aplica esta funcionalidad en la aplicación y se ejecuta en intervalos de 10 segundos. Cuando detecta que el navegador no tiene internet, muestra una alerta y desactiva el intervalo para no seguir molestando al usuario.

***Add2Home***

Add2Home[102] es una librería personalizable que detecta el terminal con el que se accede a la aplicación web para personalizar el mensaje y posición del mismo. Si se trata de un smartphone iOS, desplegará un globo informativo que sugerirá al cliente añadir la aplicación a la home screen. De esta forma, se anima al cliente a

[102] Add2Home para iOS: http://cubiq.org/add-to-home-screen (6 jun, 2013)

instalar la aplicación en la home screen y lograr una experiencia aún más similar a la de una aplicación nativa.

## 8.3. Prueba de concepto

Todas las pruebas iniciales se hacen a través del navegador. El navegador que se utiliza de base es Google Chrome por sus herramientas de desarrollador, por la experiencia adquirida con uso diario y previo conocimiento del mismo. Una vez se logran los primeros resultados satisfactorios en este navegador, se prueba con Mozilla Firefox y se continúa con Safari y Opera.

Al considerarse que la aplicación funciona correctamente en los diferentes navegadores mencionados, se abre la máquina virtual a la red local para poder realizar pruebas con los dispositivos móviles disponibles. En este caso, un iPhone 4S.

Debido a que no se tiene disponible un terminal Android, se hará uso del simulador online de dispositivos Android que pone a disposición Manymo[103] en un apartado de su aplicación web.

Se realizan pruebas de navegabilidad, almacenamiento local, desconexión de la red para comprobar el funcionamiento del cacheo de la aplicación. También se prueba a borrar toda la información de la caché y memoria para volver a comenzar de nuevo en repetidas ocasiones.

Las primeras pruebas, hasta lograr la primera versión estable, se realizan en local conectando el cliente a la IP de una red local de una máquina virtual en el propio ordenador personal. Una vez lograda la primera versión estable, se prueba su funcionamiento repitiendo las pruebas en el servidor cloud de producción en Amazon.

[103] Manymo: Simulador de Android online: https://www.manymo.com/emulators (9 jun, 2013)

## 8.4. Conclusiones

En este apartado se han analizado varias de las nuevas funcionalidades propuestas en HTML5, como el almacenamiento local, el cacheo de ficheros y el geoposicionamiento del terminal, entre otros.

Usar HTML5 en un entorno de producción se contempla arriesgado puesto que aún no es un estándar cerrado y las especificaciones pueden ser alteradas y modificadas con las consecuentes actualizaciones de los navegadores. Por tanto, una simple actualización del navegador puede variar la forma de utilizar alguna de las funcionalidades de HTML5, con graves resultados en el funcionamiento de la aplicación desarrollada. Sin embargo, es una tecnología muy avanzada que merece ser estudiada para adoptarla de forma rápida en el momento que se dé por concluida.

HTML5 proporciona numerosas funcionalidades que hacen que las aplicaciones web se asemejen mucho a las aplicaciones nativas y que, hasta el día de hoy, eran complejas y tediosas de desarrollar. Un único desarrollo de una aplicación web podrá ser ejecutado en cualquier navegador actualizado sin necesidad de descargarla desde ninguna tienda de aplicaciones, haciendo de ella una aplicación multi-plataforma.

Para determinadas aplicaciones de uso intensivo del hardware y gráficos de un dispositivo, el desarrollo nativo siempre encajará mejor. Pero la gran mayoría de las aplicaciones de consumo actuales, podrían ser migradas a un entorno HTML5 ahorrando muchas horas de desarrollo y bajando los precios para los clientes tanto por ser solo necesario un único desarrollo, como por utilizar una tecnología flexible, accesible y cercana.

Además, aquellas aplicaciones que requieran operaciones de cómputo potentes y que actualmente lo hagan en el cliente, podrían ejecutarse en la nube, devolviendo posteriormente los resultados. De esta forma se lograría que aplicaciones de mucho coste computacional, puedan ejecutarse en terminales de gama baja.

Con la estandarización de la especificación de HTML5, otra ventaja de las aplicaciones web es la poca fragmentación a la que estarán sometidas. Un desarrollo nativo siempre estará sujeto a los cambios que se realicen en las APIs del SDK de cada plataforma, lo que conlleva que una actualización del sistema operativo puede trastocar el funcionamiento de la aplicación y requiera actualizaciones para compatibilizarla (multiplicando este proceso por cada plataforma).

Los resultados que hoy en día puede dar HTML5, teniendo en cuenta que aún es un estándar sin cerrar, combinado con el resto de tecnologías utilizadas en este trabajo, son satisfactorios y pueden llegar a ser muy próximos a lo que sería una solución nativa.

# 9. Conclusiones

Este trabajo ha realizado un análisis de una serie de herramientas y tecnologías emergentes que abarca el ámbito del desarrollo de aplicaciones multiplataforma para smartphones, alternativas al uso de soluciones nativas, y la explotación de los servicios de una aplicación servidor orientada al tratamiento intensivo de datos con tecnologías preparadas para el escalado.

Además, las tecnologías analizadas se han aplicado en el desarrollo de una prueba de concepto y, con ella, se ha demostrado y comprobado que es factible obtener resultados satisfactorios con dichas tecnologías y lograr un comportamiento muy similar a la solución nativa.

La implementación del lado del servidor con la plataforma Node.js, basada en el lenguaje javascript, se considera prometedora para trabajos futuros. La instalación por defecto es ligera, muy sencilla y simple. Con la gran comunidad que le rodea, existen numerosos módulos para adaptar la solución a los requisitos del producto, Node.js es el núcleo del servidor. De esta forma, se logra un resultado ajustado sin tener que perder recursos o mover un sistema más pesado con características y funcionalidades innecesarias.

Después de realizar la implantación en el servidor de producción, se ha comprobado que, pese a ser un sistema relativamente nuevo, Node.js es un sistema estable al no haber sufrido ninguna caída y demostrado un funcionamiento correcto durante el período de pruebas. Relacionado con el material cursado en el curso de Computación en Red, el servidor está alojado en los servicios cloud de Amazon Web Services.

El sistema de gestión de bases de datos, utilizado para el almacenamiento de información, ha sido MongoDB, de orientación documental y del que ya se habló en el curso de Desarrollo de Sistemas Innovadores de Gestión de Datos. Su funcionamiento ha sido muy positivo, pudiendo implementar todos los requisitos

planteados. Las operaciones de búsqueda en función de un índice de tipo coordenada han sido satisfactorias y, además, MongoDB permite realizarlas de diferentes formas, ofreciendo operadores alternativos para ajustar las búsquedas a los requisitos de la aplicación o lograr resultados diferentes.

La facilidad de uso y la rápida adaptación a MongoDB hacen de él un SGBD a tener muy en cuenta en el desarrollo de aplicaciones de servidor basadas en el tratamiento avanzado de datos y lectura-escrituras intensivas. La extensa comunidad, la gran documentación de que dispone y el continuo mantenimiento del sistema con actualizaciones y mejoras periódicas, hace que el sistema evolucione favorablemente.

Respecto al lado del cliente, el análisis ha probado que es posible sustituir el desarrollo de aplicaciones nativas por aplicaciones basadas en la web haciendo uso de estándar HTML5 para el cacheo, almacenamiento local y geo-posicionamiento del terminal combinadas con las tecnologías CSS3, javascript y frameworks que hacen uso de ellas.

Se considera que la especificación de HTML5 está ya preparada para implementar un gran porcentaje de las aplicaciones migrables. Sin embargo, se ha descartado la continuidad de Web SQL Storage con lo que la alternativa a un almacenamiento local avanzado recae en IndexedDB que, a fecha de este estudio, no está aún implementado en todos los navegadores. Pese a esto, la comunidad ha desarrollado un driver para adaptar IndexedDB en los navegadores no compatibles que es completamente funcional y se ha utilizado con resultados positivos en este trabajo.

Por este motivo, se recomienda esperar a que el estándar de HTML5 se cierre para que todos los navegadores se estabilicen e implementen las funcionalidades que la especificación define. De esta forma se logrará que la aplicación funciona correctamente, independientemente del navegador que se utilice. Aún así, el trabajo de los responsables de los navegadores punteros para adaptar sus programas a HTML5 ya está muy avanzado.

Para las pantallas y la transición entre ellas, se ha utilizado jQuery Mobile, que facilita la implementación de aplicaciones móviles web presentando un framework con un gran número de funcionalidades que imitan el comportamiento de una aplicación móvil nativa. El resultado ha sido positivo.

Se considera que se ha profundizado en los conceptos presentados en el curso de Desarrollo de Software Seguro en Entornos Web y de Sistemas Ubicuos al analizar tecnologías relacionadas con las aplicaciones web cliente-servidor independientes del dispositivo y que consumen información relacionadas con el entorno.

Finalmente, se concluye que la integración de las tecnologías emergentes estudiadas en este trabajo fin de máster, para el desarrollo de aplicaciones cliente-servidor multiplataforma es una alternativa a tener en cuenta frente al desarrollo nativo. Pero es necesario mantenerse alerta debido a que es un sector que evoluciona con cierta velocidad, más tratándose de tecnologías innovadores como las estudiadas en este trabajo, y surgen nuevas versiones constantemente incluyendo nuevas funcionalidades y corrigiendo comportamientos incorrectos.

# 10. Trabajo futuro

Se ha implementando un subconjunto del alcance de una prueba de concepto en este trabajo fin de máster. Como trabajo futuro se tratará completar la implementación de la prueba de concepto con todas las funcionalidades que se definieron en el alcance. Así, se implementará un sistema de registro de usuarios a través de APIs de identificación de personas como Facebook, Twitter o Amazon y se estudiará sobre las nubes de relaciones de personas, su legalidad y seguridad de almacenamiento de dichos datos.

En el futuro se intentará que, los usuarios de la aplicación puedan guardar sus configuraciones y datos de establecimientos en la nube posibilitando mantener dos aplicaciones sincronizadas en diferentes terminales.

También se permitiría al cliente interactuar más con los establecimientos: marcar lugares visitados, favoritos o pendientes; permitir ordenar las listas o los establecimientos de éstas; crear rutas; o compartir con otros usuarios registrados estas acciones.

Con el fin de lograr una apariencia más nativa, se hará un esfuerzo mayor para lograr un cliente aún más asíncrono y se transformarán todas las solicitudes de datos al servidor a peticiones AJAX asíncronas con el fin de descargar únicamente datos y que sea el cliente quien los trate y los *renderice* en pantalla. Con esto último se lograría cachear toda la aplicación y el único trasiego de información entre cliente-servidor sería de datos, logrando aún una experiencia más nativa.

Será necesario investigar cómo realizar solicitudes al servidor sin cargar parámetros sobre una URL para poder cachear los ficheros. Como ya se ha mencionado en la memoria, los ficheros que se listan en el manifest App Caché sólo cachea aquellos ficheros que coincidan exactamente con el nombre. Los diferentes parámetros sobre una URL se consideran como ficheros diferentes.

Como línea más prometedora, se contempla estudiar fuentes de datos de carácter social y hacer uso de iniciativas como el Linked Open Data con el fin de poder ofrecer datos especializados proporcionados por entidades públicas y mejorar la masa de datos de la que se dispone.

Además, comprobado el buen funcionamiento de MongoDB en el trabajo, se contemplará realizar un estudio distribuyendo las colecciones de datos en diferentes servidores para ver el comportamiento de replicación, distribución y efectividad con la que funcionaría el SGBD.

Otra opción pone el foco, no solo el sistema de gestión de bases de datos, sino en analizar el comportamiento de Node.js haciendo que éste se ejecute en varias máquinas pudiendo realizar pruebas con el balanceado de carga.

# 11. Referencias

[alme2012]: Eduardo José De Almeida. *Documento de memoria: GeoFriends Windows Phone (2012). Proyecto Fin de Carrera de Ingeniería en Informática. Facultad de Informática de Donostia – San Sebastián. UPV-EHU.*

[andk2012]: Karl Andersson, Dan Johansson. *Mobile e-Services Using HTML5 (2012). IEEE Workshop On User Mobility and Vehicular Networks, 10.1109/LCNW.2012.6424068*

[bert1989]: Tim Berners-Lee. *Information Management: A Proposal (1989). W3C* http://www.w3.org/History/1989/proposal.html

[bert2009]: Tim Berners-Lee. *Linked Data Design Issues (2009). W3C* http://www.w3.org/DesignIssues/LinkedData.html

[bree2000]: Eric Brewer. *Towards Robust Distributed System (2000). ACM PODC '20, 10.1145/343477.343502*

[boia2012]: Alexandru Boicea, Florin Radulescu, Laura Ioana Agapin. *MongoDB vs Oracle - database comparison (2012).* IEEE Emerging Intelligent Data and Web Technologies (EIDWT) 2012*, 10.1109/EIDWT.2012.32*

[gari2012]: Iván García. *Documento de memoria: GeoFriends Android (2012). Proyecto Fin de Carrera de Ingeniería en Informática. Facultad de Informática de Donostia – San Sebastián. UPV-EHU.*

[gawr2012]: Rachel Gawley, Jonathan Barr, Michael Barr. *Native to HTML5: A Real-World Mobile Application Case Study (2012). Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2012, 10.1007/978-3-642-32320-1_13*

[ghea2010]: Antonio Ghezzi, Raffaello Balocco, Andrea Rangone. *How a new distribution paradigm changes the core resources, competences and*

*capabilities endowment: the case of Mobile Application Stores (2010). Mobile Business and 2010 Ninth Global Mobility Roundtable (ICMB-GMR), IEEE International Conference, 10.1109/ICMB-GMR.2010.72*

[gug2012]: Genqiang Gu, Qingchun Li, Xiaolong Wen, Yun Gao, Xuejie Zhang. *An Overview of Newly Open-Source Cloud Storage Platforms (2012). Granular Computing (GrC) 2012, IEEE International Conference, 10.1109/GrC.2012.6468625*

[groa2005]: Alan Grosskurth, Michael W. Godfrey. *A Reference Architecture for Web Browsers (2005). Software Maintenance, 2005. ICSM'05, IEEE International Conference, 10.1109/ICSM.2005.13*

[haum2010]: Michael Hausenblas, Marcel Karnstedt. *Understanding Linked Open Data as a Web-Scale Database (2010). IEEE Advances in Databases Knowledge and Data Applications (DBKDA), 10.1109/DBKDA.2010.23*

[juna2013]: Antero Juntunen, Eetu Jalonen, Sakari Luukkainen. *HTML 5 in Mobile Devices – Drivers and Restraints (2013). IEEE System Sciences (HICSS), 10.1109/HICSS.2013.253*

[kalp2011]: Panagiotis Kalagiakos, Panagiotis Karampelas. *Cloud Computing learning (2011). IEEE International Conference, Application of Information and Communication Technologies (AICT), 10.1109/ICAICT.2011.6110925*

[lita2010]: Adam Lith, Jakob Mattson. *Investigating storage solutions for large data: A comparison of well performing and scalable data storage solutions for real time extraction and batch insertion of data (2010)*

[ojaa2012]: Andrés Ojamaa, Karl Düüna. *Security Assessment of Node.js Platform (2012). IEEE Internet Technology And Secured Transactions (ITST)*

[prid2008]: Dan Pritchett. *BASE: An Acid Alternative (2008). ACM Queue, 1542-7730/08/0500*

[sind2012]: David Sin, Erin Lawson, Krishnan Kannoorpatti. *Mobile web apps – the non-programmer's alternative to native applications (2012). Human System Interactions (HSI), IEEE International Conference, 10.1109/HSI.2012.11*

[trak2012]: Kim W. Tracy. *Mobile application development experiences on Apple's iOS and Android OS (2012). Potentials, IEEE, 10.1109/MPOT.2011.2182571*

[vali2012]: Ibai Valencia. *Documento de memoria: GeoFriends Apple iOS (2012). Proyecto Fin de Carrera de Ingeniería en Informática. Facultad de Informática de Donostia – San Sebastián. UPV-EHU.*

[wang2012]: Guoxi Wang, Jianfeng Tang. *The NoSQL Principles and Basic Application of Cassandra Model (2012). IEEE Computer Science & Service System (CSSS), 10.1109/CSSS.2012.336*

[weaa1998]: Alfred C. Weaver. *Profiting from the Internet and the World Wide Web (1998). IEEE Industrial Electronics Society, 1998. IECON '98, 10.1109/IECON.1998.723929*