UNIVERSIDAD CARLOS III DE MADRID – ESCUELA POLITÉCNINCA SUPERIOR

Implementación de un gadget OpenSocial para Mensajería Instantánea

Proyecto Fin de Carrera

Autor: Marcos del Bas Junquera 100072617

Tutor: Mario Muñoz Organero

Departamento: Ingeniería Telemática

Gracias

A mi tutor Mario y a Iván por toda la ayuda y apoyo prestado durante estos 8 meses.

A Riki y Agui. Sin vosotros el paso por la universidad hubiera sido mucho más duro. Gracias por tantos días de tupper.

A Josan y Paloma, por el apoyo incondicional cada día. Por la comprensión y siempre tener palabras de ánimo.

A mis hermanos Rober y Edu, por ser un espejo en el que reflejarme cada día y mostrarme el camino.

A Paula, porque son casi 6 años admirándote como persona y como amiga. Las cosas van a salir bien.

A mis padres, por la educación recibida y por el apoyo durante estos años. Sin vosotros éste momento no hubiera llegado.

Contenido

Índice de Ilustraciones

Índice de tablas

Implementación de un gadget OpenSocial para mensajería instantánea

1. Introducción y Objetivos

1.1 Introducción

El presente proyecto pretende abordar un problema básico dentro del mundo médico cómo es la comunicación instantánea mediante la mensajería en un entorno social.

La aplicación implementada permite a los usuarios de una red social comunicarse entre sí añadiendo un gadget OpenSocial de mensajería instantánea a su red social desde cualquier dispositivo que permita ejecutar un navegador. Este gadget OpenSocial se integra en la red social y aprovecha sus características sociales para extraer listas de amigos y usuarios y de esta manera poder conectarlos entre sí mediante el protocolo XMPP utilizando un servidor Openfire.

Aprovechando las ventajas que ofrece utilizar un protocolo de comunicación potente como XMPP, el gadget nos ofrecerá servicios más allá de la propia mensajería instantánea que nos permiten interaccionar con los otros usuarios de la red social. La aplicación incorpora también un servicio de presencia donde el usuario podrá establecer su disponibilidad en cualquier momento y un servicio de suscripción donde poder establecer nuevas conexiones con otros integrantes de la red social y que a su vez nos permite conocer la presencia de estos usuarios en la aplicación.

Con todos estos ingredientes, se pretende marcar un punto de apoyo que mejore la calidad y la productividad del trabajo de los profesionales sanitarios.

1.2 Objetivos

El objetivo principal que persigue este proyecto es el de habilitar una herramienta software que facilite la comunicación entre los agentes del mundos hospitalario y que abra la puerta para mejorar la asistencia sanitaria en pacientes externos a través del control de los mismos a través de una red social.

A partir de ahí se han marcado algunos objetivos secundarios en el proyecto: cómo es que la herramienta pueda ser utilizada desde cualquier dispositivo portátil y que la herramienta sea de fácil uso y manejo.

1.3 Estructura del documento

Este documento contará con siete capítulos en los que se tratará de reflejar todo el trabajo realizado a lo largo de la duración del proyecto. De igual manera, el documento se estructurará tratando de mostrar las distintas fases realizadas en el proyecto, dando mayor importancia a los eventos acaecidos con mayor relevancia.

En el primer capítulo, a excepción de este subcapítulo, se tratará de introducir de una manera breve y fácilmente entendible cuales son los objetivos de este proyecto y cómo ha sido organizado para la consecución de dichos objetivos.

A lo largo del segundo capítulo se realizará un estudio de la tecnología necesaria para la implementación de este proyecto, detallando aquellas opciones que se han barajado como soluciones para el problema que se plantea y profundizando en la opción finalmente elegida.

En el tercer capítulo, se realizará una descripción completa de la tecnología y la arquitectura que se ha utilizado para la implementación del proyecto, haciendo un repaso por las características esenciales de cada elemento que conforma el sistema implementado y dando mayor importancia a aquellos servicios que hemos utilizado para dar solución a los requisitos planteados. Se explicará en detalle cómo ha sido el proceso seguido durante cada una de las fases de iteración del proyecto donde se acumula la mayor parte del trabajo. También se detallarán los requisitos finales y los casos de uso de la aplicación.

El capítulo cuarto aborda la descripción de las pruebas que se le han realizado al software implementado para que cumpla con los estándares de los lenguajes de programación utilizados, así como el detalle de los resultados obtenidos de estas pruebas.

Se detallarán una evaluación del cumplimiento de los requisitos y de los objetivos marcados al inicio del proyecto en el quinto capítulo.

En el sexto capítulo quedarán reflejadas todas las aportaciones personales que se han incluido dentro del proyecto así como una valoración realizada a término del proyecto.

En el séptimo capítulo se plantearán las conclusiones que se han extraído a lo largo del proyecto y se tratarán de desarrollar teniendo en cuenta los medios empleados y los objetivos marcados.

Finalmente, se añadirán las referencias, un glosario aclaratorio y aquellos documentos anexos relevantes para la comprensión del proyecto.

1.4 Planificación del proyecto

Para el desarrollo del proyecto se ha seguido un modelo iterativo e incremental al que le hemos añadido una etapa inicial en la que se ha estudiado el caso y las posibilidades del proyecto y una etapa final en la que se ha tratado de mantener la aplicación y de solucionar pequeños errores de programación y mejoras sencillas que se pudieran hacer en la aplicación.

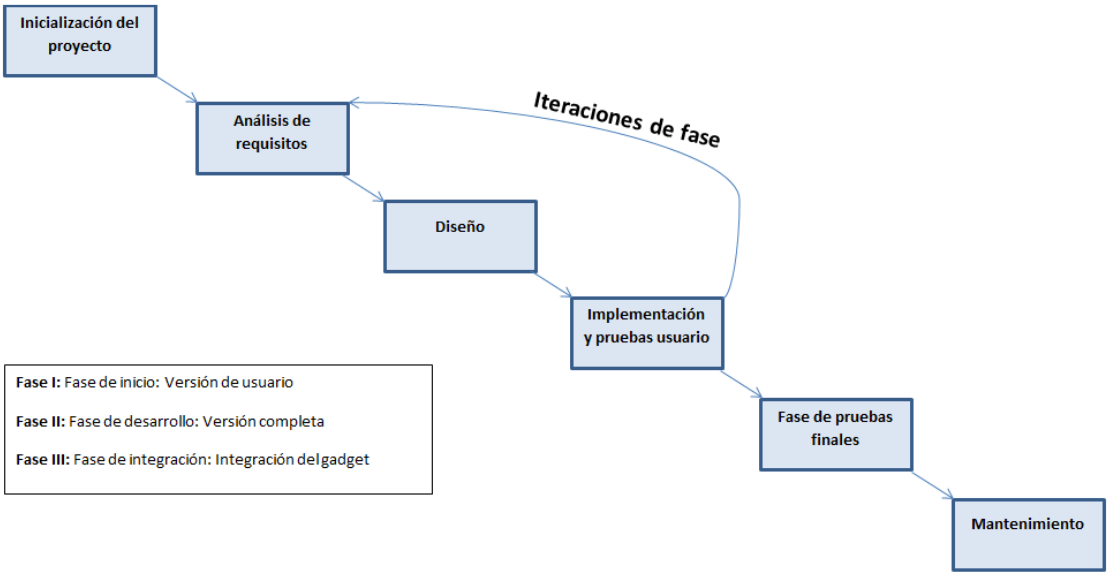


Ilustración 1 - Fases del proyecto

Como se puede ver en la imagen, se han realizado tres iteraciones dentro del modelo cuyo trabajo y resultados se detallarán en el capítulo de desarrollo y que resumimos a continuación:

En la **fase de inicialización** del proyecto se plantea el proyecto y los objetivos principales del mismo. Se buscan las tecnologías más potentes y adecuadas para la realización del mismo y se estudian estas soluciones de manera teórica para facilitar las fases de implementación.

En una primera **fase de inicio** se busca realizar una versión de la aplicación con funcionalidades reducidas que sirva como modelo de base para la futura aplicación. Esta fase también sirve para asentar los conocimientos teóricos que se habían adquirido en la

fase de inicialización. Se fijan unos requisitos básicos que sirvan como punto de partida para las acciones a realizar, se diseña una aplicación muy sencilla con las funcionalidades mínimas para poder establecer pruebas sobre los distintos elementos y se implementa esta versión de la aplicación basándose en los conocimientos adquiridos en la etapa anterior y en el diseño realizado para esta iteración. Finalmente se realizan sencillas pruebas de usuario para comprobar que la aplicación ejecuta correctamente y cumple con los requisitos que se han fijado para dicha iteración.

En la segunda fase iterativa, la **fase de desarrollo** se trata de mejorar la primera versión de la aplicación para hacerla más atractiva para los usuarios, de mejor manejo y mayor funcionalidad. Además, se incorporan nuevos servicios como el de presencia y añadir usuarios a la aplicación. Se añaden algunos requisitos nuevos que reflejen la identidad de esta nueva versión de la aplicación y se diseña una interfaz gráfica más clásica, intuitiva y atractiva para los usuarios. Para la implementación se incorporan nuevos elementos gráficos basados en las librerías de jQuery UI y finalmente se vuelven a realizar las mismas pruebas de usuarios que en la etapa anterior, de nuevo buscando la validación de requisitos y la experiencia de usuario.

Para finalizar con las fases iterativas, en la **fase de integración** se trata de integrar la segunda versión de la aplicación realizada en la etapa anterior dentro de la red social escogida como contenedor para el gadget OpenSocial. De esta manera, se fijan los requisitos finales de la aplicación teniendo en cuenta la presencia de la red social, se diseñan los cambios a realizar para que la integración sea lo más completa posible y se implementan estos cambios gracias al API de OpenSocial. Por último se realizan pruebas con usuarios de la red social para comprobar la funcionalidad de la aplicación y su nivel de integración con la red social.

Tras las etapas iterativas, a la estructura del proyecto se le ha añadido una **fase de pruebas final** en la que se ha comprobado que se cumplan todos los requisitos en materias de estándares de las tecnologías aplicadas y en conceptos necesarios como son la seguridad o la funcionalidad.

Para finalizar, se añade una última **etapa de mantenimiento** en la que se busca una versión más estable de la aplicación a través de la solución de pequeños bugs encontrados.

El trabajo realizado a lo largo de todo el ciclo de vida del proyecto, queda reflejado en una línea temporal en el siguiente diagrama de Gantt.

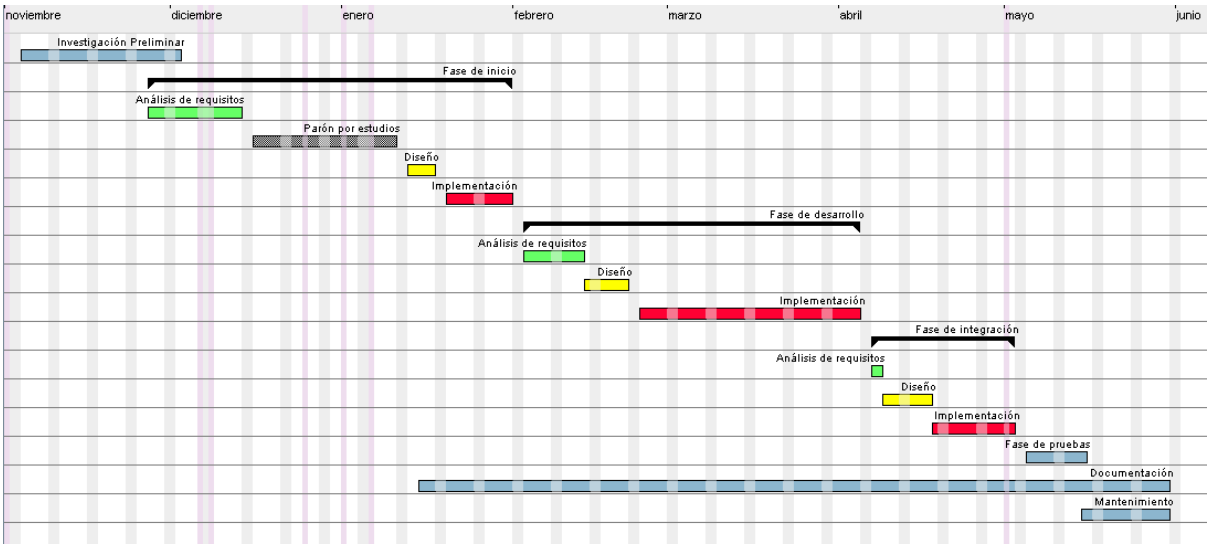


Ilustración 2 - Diagrama de Gantt para planificación del proyecto

A su vez, podemos ver las fechas de inicio y final de cada fase en la siguiente tabla:

Nombre	Fecha de inicio	Fecha de fin
Investigación Preliminar	4/11/13	2/12/13
Fase de inicio	27/11/13	31/01/14
Análisis de requisitos	27/11/13	13/12/13
Parón por estudios	16/12/13	10/01/14
Diseño	13/01/14	17/01/14
Implementación	20/01/14	31/01/14
Fase de desarrollo	3/02/14	4/04/14
Análisis de requisitos	3/02/14	13/02/14
Diseño	14/02/14	21/02/14
Implementación	24/02/14	4/04/14
Fase de integración	7/04/14	2/05/14
Análisis de requisitos	7/04/14	8/04/14
Diseño	9/04/14	17/04/14
Implementación	18/04/14	2/05/14
Fase de pruebas	5/05/14	15/05/14
Documentación	15/01/14	30/05/14
Mantenimiento	15/05/14	30/05/14

Tabla 1 - Fases del proyecto

1.5 Medios empleados

1.5.1 Recursos Hardware

Para la realización de este proyecto, se ha contado con un servidor de pruebas dentro de la universidad en el cual se han instalado tanto el servidor Openfire como el servidor Apache Rave, así como con un ordenador portátil para la planificación del proyecto, diseño de la aplicación e implementación del código .

Además del servidor Openfire instalado en la universidad en la dirección http://163.117.141.214:7070, se ha contado con otro servidor de pruebas instalado en el portátil. Igualmente, a parte del servidor Apache Rave instalado en la universidad en la dirección http://163.117.141.214:8080, se ha utilizado otro servidor auxiliar instalado en el portátil.

1.5.2 Recursos Software

En cuanto a los recursos software utilizados, para la planificación del proyecto se han utilizado el programa *GanttProject* y *Microsoft Excel 2010.*

Para la implementación y el desarrollo del software se ha utilizado el entorno de desarrollo integrado libre *NetBeans* y para las pruebas y ejecución del programa cliente se han utilizado los navegadores *Google Chrome* y *Mozilla Firefox*.

Finalmente, para la gestión de usuarios de la red XMPP se ha utilizado la consola de administración del servidor Openfire, instalada en el puerto 9090 del servidor de pruebas ubicado en la universidad.

1.5.3 Recursos humanos

En relación a los recursos humanos que han participado en el desarrollo del proyecto, se ha contado con un recurso a tiempo parcial en las primeras fases del proyecto y a tiempo completo a partir de la fase de desarrollo del proyecto.

También se ha contado con el apoyo en la instalación del servidor de la universidad y las pruebas del software en el mismo servidor de un recurso a tiempo parcial durante algunas fases del proyecto.

2. Estado del arte

2.1 Introducción

En este capítulo se tratará de discutir las diferentes alternativas tecnológicas que se han ido proponiendo ante los problemas surgidos durante el planteamiento tecnológico para desarrollar un gadget de mensajería instantánea.

2.2 Historia de la mensajería instantánea

Aunque el verdadero boom llegara a partir de los años noventa, la historia de la mensajería instantánea (IM), se extiende desde hace cinco décadas, en las que los cambios y las innovaciones han sido constantes.

1960s

El concepto de IM tiene su origen a mediados de esta década. Los sistemas operativos multi-usuarios tales como *Compatible Time-Sharing System* (CTSS), que fue creado en el centro de informática del MIT en 1961, permitían hasta a 30 usuarios compartir el mismo sistema e intercambiar mensajes entre ellos. El sistema, que en la actualidad se acerca más a la idea que tenemos sobre el email, logró que cientos de usuarios del MIT se registraran.

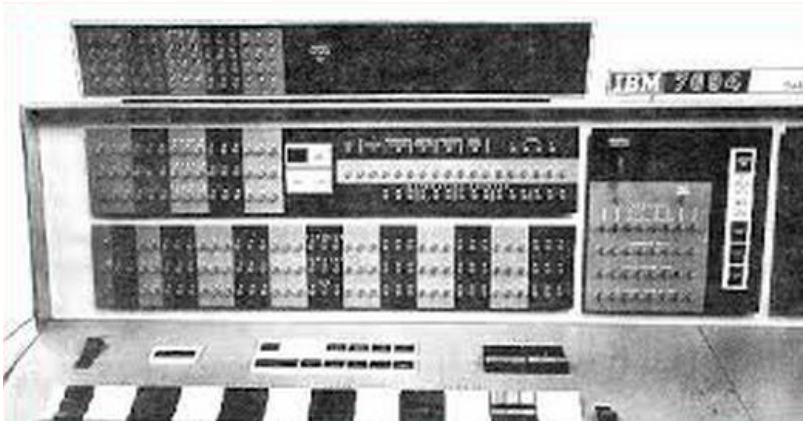


Ilustración 3 - CTSS usaba un sistema IBM 7094 modificado

1970s

En esta época, los programadores comenzaron a trabajar en el protocolo peer-to-peer (P2P), permitiendo a los investigadores establecer comunicación entre usuarios de un mismo sistema.

1980s

Se desarrolla en el MIT un sistema basado en UNIX para enviar mensajes entre usuarios. Se trata del *Zephyr Notification Service*, que aún se utiliza en algunas instituciones.

También en esta época se desarrolla el sistema BBS (*bulletin board system*), un sistema que permite a través de un terminal subir y descargar software, así como intercambiar mensajes entre los distintos usuarios.

En 1982, *Commodore International* lanza al mercado el ordenador personal *Commodore 64.* Este PC, incluía un sistema para dar un servicio de mensajería instantánea llamado *Quantum Link* (Q-Link) que permitía a los usuarios, pagando una cuota mensual enviar mensajes a otros usuarios vía modem.

Ilustración 4 - Commodore 64

Un año más tarde, en 1983 un estudiante de secundaria de Washington llamado Mark Jenks crea "Talk", un sistema que permite a estudiantes y profesores comunicarse a través de placas digitales desarrolladas para la mensajería y un sistema parecido a los buzones de correos del email.

A finales de esta década, en 1988, IRC es creado por Jarkko Oikarinen con el motivo de remplazar un sistema multiusuario de un BBS instalado en Finlandia. Este sistema ganó en popularidad a partir de su uso para evitar la censura en un golpe de estado fallido en la Unión Soviética en 1991.

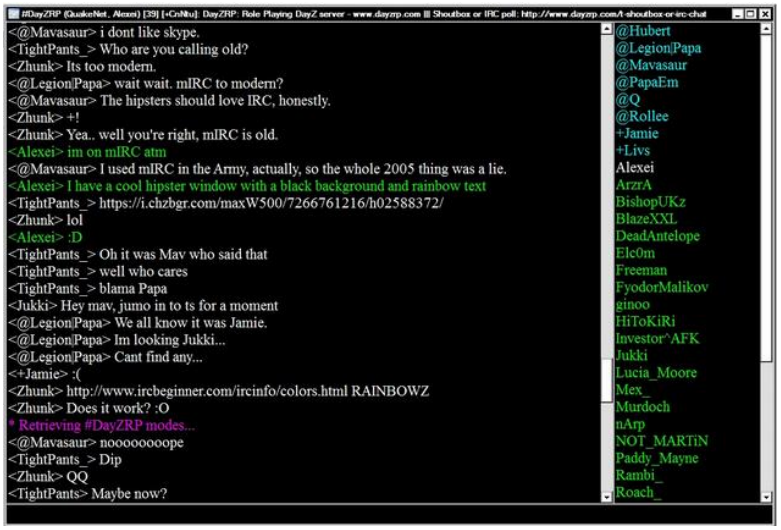


Ilustración 5 - Interfaz de IRC

La popularización del chat

En la década de los noventa, el uso de Internet se comenzó a popularizar entre los usuarios de los ordenadores personales. Este hecho ayudaría a que muchas compañías comenzaran a crear mercados de servicios dentro de Internet.

En noviembre de 1996, una compañía israelita llamada *Mirabilis* lanza ICQ. Se trataba del primer chat al uso y popularizado que alcanzó un número muy importante de usuarios en Internet. La antigua Quantum Link, que en 1991 cambia su nombre por America Online (AOL), adquiere en 1998 *Mirabilis*.

Un año antes, en 1997, AOL había lanzado al mercado un producto que revolucionó el mercado tal y como se conocía. AIM contaba con salas de chat, listas de amigos, iconos, perfiles de usuario y mensajes de ausencia. En 2005 AIM dominaría el mercado del chat con 53 millones de usuarios.

Ilustración 6 - Interfaz de chat AIM en una de sus primeras versiones

En 1998, Yahoo, lanza *Yahoo! Messenger*. Para utilizarlo, tan sólo necesitabas de un identificador de Yahoo. Al igual que AIM, este chat contaba también con un servicio de salas, mensajes de ausencia y listas.

Pero fue un año después, ya en 1999, cuando Microsoft lanzó al mercado el producto más popularizado en España y uno de los más utilizados en todo el mundo. *MSN Messenger* dominaría el mercado durante los primeros años del siglo XXI. En 2009, *MSN Messenger* alcanzaría la cifra de 330 millones de usuarios activos.

Ilustración 7 - Primera versión MSN Messenger

Los primeros años del siglo XXI

MSN Messenger se convertiría en el gran dominador de esta época, sin embargo, otras compañías comenzarían a desarrollar otros protocolos y sistemas que acabarían dominando el mercado.

En el año 2000, el cliente multi-protocolo Jabber, basado en XMPP, es lanzado el mercado, permitiendo a los usuarios acceder a todas las listas de amigos de sus aplicaciones IM desde una sola aplicación. Ese mismo año, aparecería *Trillian*, también un cliente multi-protocolo que permitía a los usuarios hablar con sus amigos de AIM, ICQ y MSN a la vez.

Ilustración 8 - Spark, uno de los clientes Jabber

Skype fue fundada en el año 2003, y desde un primer momento ofrecía a los usuarios, además de conexión de videoconferencia, mensajería instantánea y servicio de comunicación por voz.

La aparición de las RRSS

Las redes sociales marcarían el cambio dentro de la historia de la mensajería instantánea.

En 2005, Google lanza su servicio *Google Talk* que permitía mantener conversaciones a partir de listas de contactos creadas en GMail.

Ilustración 9 - GTalk en su versión beta

MySpace es lanzada en 2006. Para convertirse en la primera red social popularizada, necesitaba de un servicio de mensajería ofrecido desde el propio escritorio de la red social. Más adelante lo externalizaría fuera de la plataforma con el lanzamiento de *MySpaceIM*.

Finalmente, Facebook en 2008 lanzaría el *Facebook Chat*, permitiendo a los usuarios de la red social mantener conversaciones con sus amigos cuando estuvieran conectados.

Mobile Instant Messaging (MIM)

MIM es un servicio que pretende trasladar el escenario anterior de ejecución de la mensajería instantánea al mundo de los móviles.

La primera aproximación a la MIM ocurre gracias a la adaptación de clientes web al mundo de los móviles, sin embargo, estas adaptaciones no tienen mucho éxito.

En 2009, Apple lanza las notificaciones push. Esto permitía a otros usuarios conocer si estabas utilizando o no determinadas aplicaciones a través de un ping a todos los usuarios de la red. Este servicio es integrado por *Whatsapp* que lanza su versión 2.0, y en pocos días el número de usuarios de la aplicación asciende a 250.000.

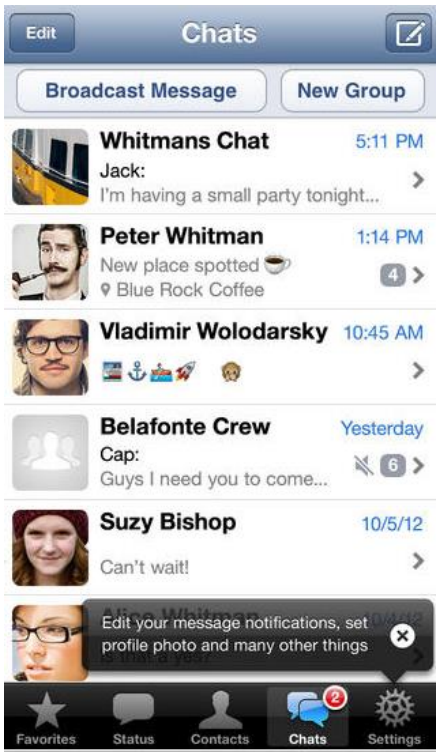


Ilustración 10 - Interfaz Whatsapp para iOS

En la actualidad, tanto compañías que ya ofrecían el servicio de IM como Facebook, Google o Yahoo, y nuevas compañías como Viber, Line o Telegram han adaptado la filosofía de Whatsapp para ofrecer servicios MIM.

2.3 Protocolos de Mensajería Instantánea

En esta sección se discutirán las distintas alternativas tecnológicas que se han barajado como base para ofrecer los servicios de mensajería instantánea y presencia.

2.3.1 Zephyr

¿Qué es Zephyr?

[13] Zephyr es un protocolo de mensajería instantánea creado en 1986 en el MIT basado en UNIX. Junto con IRC se les considera los primeros protocolos de mensajería instantánea basados en IP.

En la actualidad Zephyr es un protocolo usado tan sólo en el ámbito universitario: tan solo universidad como el MIT o la Universidad de Brown se resisten a que sea remplazado por otros protocolos más modernos como XMPP.

¿Cómo opera Zephyr?

Este protocolo utiliza como capa de transporte UDP. Manda datagramas desde los puertos 2102, 2103 y 2104.

Es incompatible con la mayoría de routers NAT, ya que no sería capaz de acceder a la IP externa de la red y por lo tanto los paquetes nunca llegarían a su destino.

Características de Zephyr

Zephyr es un protocolo de mensajería instantánea muy robusto que tan solo permite dos servicios: mensajería instantánea y multi conferencias.

- **Clases:** El tema más interesante acerca de este protocolo son las llamadas *classes*. Estas clases, son una especia de sala de chat desde el que se puede intercambiar mensajes que otros sistemas. Cualquiera que esté suscrito a una clase, recibirá los mensajes de los usuarios que envíen mensajes a esa clase. La suscripción a las clases carece de seguridad, por lo que es imposible conocer cuáles son los usuarios suscritos en una clase.
- **Instancias:** Cuando se envía un mensaje a una clase, generalmente va acompañado de una instancia. Podemos pensar en la instancia como una especie de tema que acompaña al mensaje. Si no se especifica la instancia de un mensaje, se presupone que el tema es de tipo *PERSONAL*.
- **Zephyr triplets:** Todos los mensajes por lo tanto incluyen tres componentes: la clase, la instancia y el receptor del mensaje.
- **Clases comunes:** Los administradores de la red puede incluir algunas clases comunes para todos los usuarios, y de esta manera acceder a temas de ayuda o temas que resulten importantes para la comunidad.

2.3.2 Mumble

¿Qué es Mumble?

[14] Mumble es una aplicación de voz sobre IP (VoIP) multiplataforma principalmente utilizada por los desarrolladores de juegos.

Mumble es un software abierto caracterizado por servir un sonido de alta calidad y una muy baja latencia. Todas las comunicaciones son encriptadas para asegurar la privacidad de los usuarios.

¿Cómo opera Mumble?

La arquitectura de Mumble se corresponde con la clásica arquitectura cliente-servidor que permite a los distintos usuarios hablar mientras están conectados al mismo servidor.

Estos servidores pueden ser alquilados o montados directamente. Montar el servidor no es muy costoso ya que se puede descargar una herramienta llamada Murmur que instala el servidor directamente.

El servidor contiene un archivo para poder configurarlo y de esta manera administrar el servidor. Este servidor utiliza el puerto 64738 TCP y UDP.

¿Qué características tiene Mumble?

- **Canales jerarquizados:** Los servidores Murmur se organizan en torno a un canal raíz a través de un árbol jerárquico. Esto permite que usuarios conectados a un canal común creen canales más pequeños temporalmente para poder hablar con algunos usuarios en concreto.
- **Alta calidad de sonido:** Mumble utiliza el códec *Speex* para, además de codificar y decodificar, reducir el ruido y como sistema de control automático de ganancia. Mumble incorpora también un sistema de cancelación de eco y está optimizado para tener una baja latencia, lo que permite que las comunicaciones sean más rápidas y reales.
- **Capa de superposición:** Mumble incorpora una capa extra para poder mostrar qué usuario está hablando en cada momento y que permite añadir avatares a los perfiles de los usuarios.
- **Audio posicional:** Mumble es capaz de posicionar las capas de audio en los lugares en los que los personajes del juego se encuentran. De esta manera, si el usuario de un personaje a tu izquierda habla, será el canal izquierdo el que se active.
- **Aplicaciones móviles:** Existen aplicaciones desarrolladas para utilizar los servicios de Mumbles tales como Mumblefy para iOS y Plumble para Android.

2.3.3 SIMPLE

¿Qué es SIMPLE?

Las siglas SIMPLE pertenecen a Session initiation protocol for Instant Messaging and Presence Leveraging Extensions. Se trata de un protocolo basado en el protocolo de señalización SIP. Al igual que los anteriores protocolos y sistemas, se trata de código abierto.

¿Por qué se basa en SIP?

[15] SIP es un protocolo de señalización creado con la intención de convertirse en un estándar para la iniciación, modificación y finalización de las sesiones de usuario donde intervengan elementos multimedia como voz, video, realidad virtual o mensajería instantánea. Por tanto, es normal que SIMPLE esté basado en SIP.

¿Cómo funciona SIP?

SIP ha sido desarrollado como un concepto de caja de herramientas, incluyendo muchas partes de otros protocolos que ya funcionaban como de los protocolos SDP o RTP.

El protocolo SIP adopta una arquitectura cliente-servidor y es transaccional. El servidor utiliza los puertos 5060 de TCP y UDP para recibir las peticiones de los clientes. Si estas peticiones vienen encriptadas, utilizará el puerto 5061. El servidor mandará una respuesta o varias al cliente como contestación de su petición, ya sea aceptando o rechazando su propuesta.

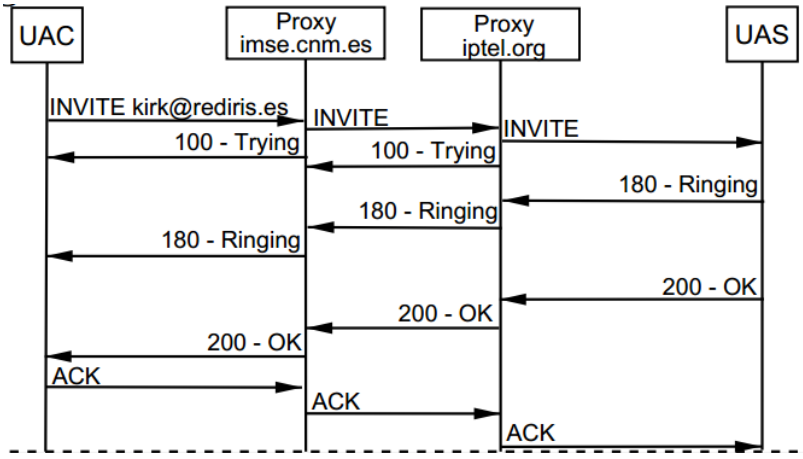


Ilustración 11 - Ejemplo de llamada en SIP

En el protocolo SIP, intervienen las siguientes partes:

- **Agentes de usuario (UA):** Pueden ser personas o software que emiten y consumen los mensajes de los protocolos. El protocolo SIP no se interesa por la interfaz de estos agentes (sean por ejemplo un móvil) si no tan sólo de los mensajes que se generan y como se ha de comportar el agente ante un determinado mensaje.

  Estos agentes de usuarios pueden tener una doble función. Pueden comportarse como clientes (UAC) o como servidores (UAS) y en función de la petición que reciban, actuarán de un lado o de otro.
- **Registrar:** el protocolo SIP utiliza unos servidores específicos para ubicar a los usuarios dentro de su red. El mecanismo utilizado funciona de la siguiente manera:

  Las direcciones SIP tienen el formato usuario@dominio. Estas direcciones se llaman direcciones lógicas y es distinta de su dirección física (dependiente de su dirección IP). Cuando un usuario inicia una sesión SIP en un terminal, lo primero que el cliente hace es enviar una petición de REGISTRAR a estos servidores de registro especificando su dirección lógica y la dirección física asociada a ella.
- **Proxys y servidores de redirección:** En la red SIP se utilizan este tipo de servidores para encaminar los mensajes de un lado a otro. Un mismo servidor puede actuar de proxy o de redirector, y lo hará en función de las necesidades del cliente. La diferencia entre una función y otra radica en que un servidor proxy encamina el mensaje hacia su destino (de un modo parecido al de los routers en las redes IP) y los servidores de redirección generan una respuesta que indica al emisor la dirección del destino o del siguiente servidor hacia el destino.
- **Servidor de localización:** Este servidor da información acerca de la posición física de un determinado usuario.

¿Qué características extras aporta SIMPLE?

Como hemos podido ver, el protocolo SIP se encarga de aportar la funcionalidad necesaria para trabajar con la mensajería instantánea. Sin embargo, hay un servicio que SIP no aporta y que es muy necesario en las aplicaciones actuales, por lo que si aparece desarrollado en SIMPLE.

SIMPLE permite a los usuarios intercambiar mensajes de presencia, indicando su disponibilidad en cada momento. De esta manera los usuarios, además de conocer si otros

se han registrado en la red SIP, pueden conocer si están aptos para hablar o por el contrario están ocupados o ausentes.

2.3.4 XMPP

¿Qué es XMPP?

[1,2,3] Extensible Messaging and Presence Protocol (XMPP) es un protocolo opensource y extensible, muy utilizado en la actualidad (por ejemplo, Google, Tuenti, WhatsApp o Facebook lo implementan o han implementado como parte de sus sistemas de chat) que permite el intercambio de datos XML.

Al igual que HTTP, XMPP es un protocolo cliente-servidor, pero difiere del mismo en que XMPP permite intercambiar información con el otro extremo de manera asíncrona. Las conexiones XMPP tienen una vida larga (se llaman HTTP-binding). En la práctica, esto implica que la información se transporta mediante operaciones "push", en vez de "pull-push".

XMPP ofrece un paso más allá en el nivel de dinamismo e interactividad de una aplicación web. Tal y como AJAX ofreció a HTTP este dinamismo, es ahora XMPP quien ofrece a AJAX la posibilidad de llevar esta indispensable cualidad a un siguiente nivel. De esta manera, XMPP trae nuevas y excelentes posibilidades de comunicación en la web, además de un menor número de paquetes intercambiados para las mismas operaciones que otras tecnologías.

XMPP, por su propia naturaleza, incluye muchas características sociales debido a la herencia recibida de otros protocolos de mensajería instantánea. Mediante XMPP podemos extraer listas de contactos de los usuarios y podemos suscribirnos a determinados usuarios para recibir sus actualizaciones de presencia y poder comunicarnos con ellos. Además dispone de más de 300 extensiones, que junto con el núcleo de su código permite construir aplicaciones de mensajería instantáneas cada vez, más sorprendentes.

¿Qué ventajas tiene XMPP?

[1] XMPP nos ofrece una serie de ventajas sobre otros protocolos.

- Más de diez años de desarrollo han permitido que XMPP se convierta en un **protocolo estable**, ampliamente desarrollado, bien testeado, muy

conocido en Internet con miles de servicios a su disposición y millones de usuarios.

- **XMPP es seguro**. Está dotado de mecanismos para la encriptación del canal y de una fuerte autenticación. Permite la encriptación end-user to end-user.
- **XMPP es descentralizado**. XMPP está desarrollado en un sistema cliente-servidor con una cantidad ilimitada de servidores. Todos los usuarios podrían montar su propio servidor y conectarse a la red XMPP a través de un DNS.
- **XMPP es extensible**. XMPP permite la transmisión de XML de un lugar a otro, por ello ha sido usado para una gran cantidad de aplicaciones como: VoIP, Instant Messaging, geolocalization, machine to machine communication, etc.
- **XMPP es escalable**. El modelo "push" de transferencia de información resuelve la gran mayoría de los problemas de escalabilidad de HTPP y su modelo "pull".
- **XMPP es una comunidad**, con una gran cantidad de desarrolladores y buen soporte.

¿Qué servicios nos ofrece XMPP?

[1] En el contexto de la aplicación que queremos desarrollar, XMPP nos ofrece los siguientes servicios centrales:

- **Encriptación del canal:** Ofrece encriptación en la comunicación cliente-servidor y en la comunicación servidor-servidor. Nos permite crear aplicaciones seguras.
- **Autenticación:** Todas las entidades de la red XMPP han sido anteriormente autenticadas por un servidor, que actúa, en este caso, como un guardián de la red.
- **Presencia:** Este servicio nos permite conocer si una entidad está o no disponible en cualquier momento de su conexión, pero además, nos permite incluir información adicional tal como la razón de su indisponibilidad, si se encuentra en una reunión, etc. Para obtener la información de presencia de un contacto, se ha de estar suscrito a dicho contacto, protegiendo la privacidad del usuario.

- **Lista de contactos:** El uso más común de este servicio, es el de otorgar al usuario la capacidad de iniciar una conversación rápidamente con cualquier otro usuario al que esté suscrito.
- **Mensajes one-to-one:** El uso tradicional en la red de este servicio es la Mensajería Instantánea. Sin embargo, en esencia se trata de un intercambio de información XML, por lo que puede ser también usado entre bots, o entre servidores, etc.
- **Mensajes multiusuario:** Este servicio nos permite utilizar salas, similares a las del mítico IRC, en las que se intercambian mensajes que pueden ser texto plano, configuración de la sala, mensajes de control de la sesión, etc.
- **Notificación:** Permite generar notificaciones y enviarlas a todas aquellas entidades suscritas.

¿Qué aplicaciones podemos implementar con XMPP?

[1]

- **Mensajería Instantánea:** Este tipo de aplicaciones se construyen, como mínimo desde la combinación de tres de los servicios de XMPP: mensajes one-to-one, lista de contactos y servicio de presencia.
- **Chat de grupo**: Los mensajes multiusuarios nos permiten construir aplicaciones más específicas en las que se intercambian mensajes en una sala grupal. Estas aplicaciones son muy usadas en cualquier tipo de industria.
- **Juegos:** la combinación de las dos aplicaciones anteriores con sus servicios nos permite construir sistemas de juego simples. También se pueden construir chats de grupo duales a cualquier juego.
- **Sistemas de control:** Los mensajes one-to-one permiten el desarrollo de aplicaciones para control remoto, como pudieran ser control de una red, control robótico, etc.
- **Geo localización:** Uno de los tipos definidos de payload para los paquetes XMPP es la geo localización. Esto nos permite construir este tipo de aplicaciones, como el seguimiento de vehículos.
- **Middleware y computación en la nube:** Aunque parezca sorprendente, la sencillez y la ligereza de las aplicaciones XMPP, permiten que muchas compañías estén apostando por desarrollar aplicaciones de este tipo.

- **Estandarización de datos:** XMPP ofrece la posibilidad de evitar el constante "polling" para actualizar los datos mediante el servicio de notificación. Ventaja que puede ser utilizada para la implementación de este tipo de aplicaciones, ahorrando de esta manera en recursos.
- **VoIP:** Muchas extensiones de XMPP se han desarrollado a partir de la aparición de GTalk permitiendo la implementación de este tipo de aplicaciones.
- **Servicios de identificación:** La utilización de identificativos estables en la red (JIDs) y el robusto servicio de autenticación, hacen posible que se use XMPP para construir servicios de identificación como OpenID o *OAuth*.

Elementos y características de XMPP

Direcciones XMPP: Una dirección XMPP (llamada Jabber ID o JID) tiene la siguiente forma: *usuario@dominio/recurso*, donde el dominio viene dado por el servidor y se refiere al área de despliegue del mismo; el usuario es un identificador único dentro de un dominio; y el recurso es el cliente desde el que se conecta un usuario (laptop, mobile, home, etc.). Cada JID tiene la característica de ser único para cada dominio.

XML stanzas: Una vez se establece una conexión TCP con el servidor, se negocia el flujo de datos XML que se va a intercambiar. Una vez se ha llevado a cabo la negociación, comienza el intercambio de stanzas. Las stanzas XML son las unidades básicas de información con significado XMPP. Existen 3 tipos de stanzas XMPP:

<message/>: Es el método básico para transportar información de un lado a otro. Este tipo de stanzas no requieren de ACK, y se usan fundamentalmente para las aplicaciones de mensajería instantánea, mensajes de grupo y notificaciones. Dentro de estas stanzas, el atributo *type* diferencia entre varios tipos: *normal, chat, groupchat, headline, error*.

Los mensajes contienen también una parte de *payload* definidas en las especificaciones del núcleo de XMPP, como <*body/*>, donde va el cuerpo del mensaje o <*subject/*>, donde se especifica el asunto del mensaje.

```
<message from="doctorMirza@hospital/office"
         to="docAllison@hospital"
         type="chat">
   <body>Let's take a break!</body>
   <subject>Query</subject>
</message>
```

<presence/>: Este tipo de stanzas sirven para notificar la disponibilidad de un usuario dentro de la red a los demás usuarios. Para que un usuario reciba una stanza de este tipo de otro usuario, este debe de estar suscrito a dicho usuario (habiéndolo autorizado este último usuario con anterioridad).

```
<presence from="doctorMirza@hospital/office">
    <show>xa</show>
    <status>Surgery time!</status>
</presence>
```

<iq/>: Muy similar al GET, POST y PUT de HTML, da una estructura para interacciones *request-response*. A diferencia de las stanzas *message*, sólo permite un *payload*, que especifica la acción a tomar por el receptor del mensaje. En el caso de estas stanzas, el atributo *type* puede tomar valores especiales: *get, set, result, error*. Se usan para conseguir rosters, suscripciones a usuarios o a salas de chat, etc.

```
<iq from="doctorMirza@hospital/office"
        id="rr8222z7"
        to="docAllison@hospital"
        type="get">
    <query xmlns="jabber:iq:roster"/>
</iq>
```

Comunicación asincrónica: Como hemos visto anteriormente, XMPP permite el envío de mensajes asíncronos que no se basan únicamente en una estructura pregunta-respuesta. En general, podemos hablar que la comunicación XMPP está dirigida por eventos que ocurren en la aplicación y funcionan como disparador de una serie de mensajes tipo "push".

Manejo de errores: XMPP asume siempre que el mensaje enviado ha llegado a su destino, a excepción de las stanzas IQ. En respuesta de una IQ-get o IQ-set siempre se recibe una IQ-result o IQ-error. Las stanzas IQ-error llevan el atributo *type* con el valor *error*, de esta manera se las reconoce, y con un elemento hijo de la stanza de tipo <error/>. Este a su vez, lleva también un atributo *type* que puede tomar los siguientes valores: *auth, cancel, continue, modify or wait*. Este atributo dirá al receptor de esta IQ-error stanza la acción a tomar ante la misma.

```
<message from="docAllison@hospital"
```

```
        to="doctorMirza@hospital"
        type="error">
    <error type="cancel">
        <service-unavailablexmlns="urn:ietf:params:xml:ns:xmpp-
        stanzas"/>
    </error>
</message>
```

2.3.5 Conclusiones

El siguiente cuadro comparativo muestra las características de los cuatro protocolos de mensajería instantánea que se han analizado.

Protocolo	Mensajes asíncronos	Seguridad en la capa de transporte	Número de contactos ilimitados	Conferencias	Audio/VoIP	Webcam/Video	Envío de archivos
Zephyr Notification Service	Sí	No	Sí	Sí	No	No	No
SIP/SIMPLE	Sí	Sí	Sí	*N/A*	Sí	Sí	Sí
Mumble	Sí	Sí	Sólo algunos bots	Sí	Sí	Sí	Sí
XMPP	Sí	Sí	Sí	Opcional	Sí	Sí	Sí

Tabla 2 - Comparativa protocolos de IM

Como se puede ver en la tabla y hemos podido comprobar a lo largo de este capítulo, los protocolos SIMPLE y XMPP parecen los más indicados para ser utilizados en la implementación de nuestra aplicación.

Ambos ofrecen todos los servicios que necesitamos o podríamos necesitar en un futuro, sin embargo hay dos puntos que les diferencias y nos hacen decantarnos por XMPP:

1) XMPP es un protocolo orientado hacia la mensajería instantánea y SIP es orientado hacia VoIP. Aunque ambos pueden realizar las funciones para ofrecer servicio de IM, XMPP está más optimizado para ello.
2) En la actualidad XMPP es un protocolo muy utilizado por grandes aplicaciones que bastante éxito, del cual se dispone de más información actualizada. El uso de SIP existe en un ámbito más empresarial.

2.4 Historia de las redes sociales

La historia de las redes sociales está muy ligada a la mensajería instantánea. De hecho, ambos inicios coinciden en la creación del sistema BBS (Bulletin Board System) en 1978.

Aunque la popularización de las redes sociales comienza a finales de la década 00s, hace más de 30 años atrás que se vienen desarrollando aplicaciones con carácter social y comunicativo.

1978

En este año, dos estudiantes de Chicago fanáticos de los ordenadores inventa el sistema BBS, mediante el cual podían informar a amigos de reuniones, hacer anuncios y compartir información a través de posts. Era el comienzo de una pequeña sociedad virtual y podemos pensar en ellos como una especie de foro actual.

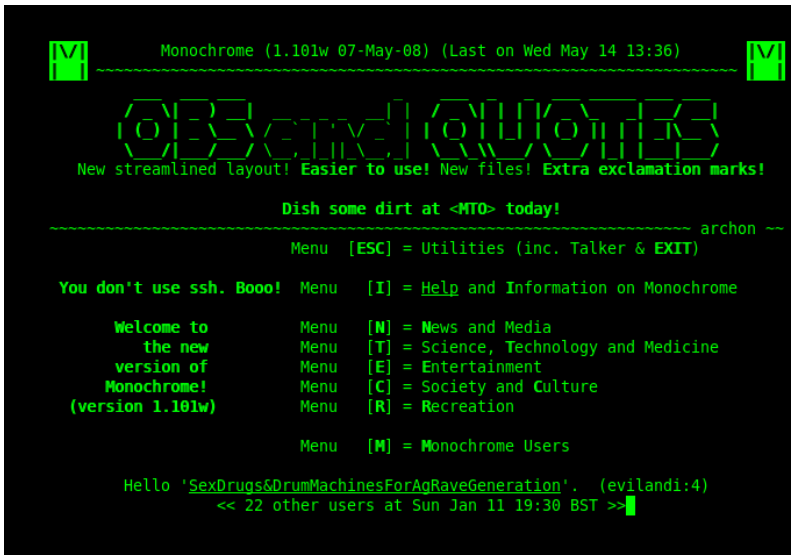


Ilustración 12 - Monochrome BBS

Un año más tarde, en 1979 esta tecnología permitiría conectar la Universidad de Duke y la universidad de Carolina del Norte.

1984

En este año comienza a desarrollarse un nuevo servicio de acceso a noticias, información del tiempo, encuestas, etc. llamado Prodigy, aunque no sería hasta 1988 que comenzaría a dar servicio en Atlanta y San Francisco.

Ilustración 13 - Interfaz del servicio Prodigy

En un primer momento, los suscriptores del servicio accedían a él a través de las antiguas redes X.25 soportando módems de hasta 1200 bits por segundo. En 1990, era el segundo servicio de la red con un mayor número de suscriptores, con 465.000.

Actualmente esta compañía es parte de la empresa de telecomunicaciones americana AT&T.

1992

En este año nace en Massachusetts de la mano de dos estudiantes y un profesor de economía el servicio Tripod. En un primer momento se trataba de una comunidad online de estudiantes y adultos. Dos años más tarde, el servicio se lanzaría a internet bajo el dominio de Tripod.com.

En estos comienzos, el sitio web trataba de dar consejos a los jóvenes que formaban parte de la comunidad ante los problemas más comunes que se les presentaban, como el primer cambio de casa e incluía un espacio para que anunciantes pudieran vender productos a sus suscriptores.

En la actualidad, este dominio sigue estando en activo bajo el soporte de Lycos, aunque el servicio que se ofrece en el mismo nada tiene que ver con el original.

1994

La compañía Beverly Hills Internet (BHI) lanza Geocities, una aplicación web que permitía a los usuarios crear su propia página personal ubicándola dentro de 6 barrios distintos. De esta manera, las páginas web de los usuarios quedaban alojadas dentro del barrio de Geocities al que pertenecían, de igual modo que hoy quedan alojados los perfiles dentro de Facebook.

Ilustración 14 - Página principal de Geocities en 1998

Los barrios se fueron ampliando y alcanzaron las seis millones de visitas mensuales a su web. Compañías más grandes como *Yahoo* comenzaron a invertir en este portal, llegando a ser comprada por esta misma compañía en 1999, dos años antes del pinchazo de la burbuja punto-com.

El éxito de nuevos servicios hico que diez años más tarde, en 1999, el servicio fuera cerrado por Yahoo al completo.

1997

En este año aparece la que por muchos es llamada como la primera red social. Sixdegrees.com era un servicio social que permitía a los usuarios mantener listas de contactos y miembros de la familia y que permitía el intercambio de mensajes entre estos.

Esta aplicación permitía a los usuarios enviar invitaciones a otras personas en la red para que se unieran a la comunidad.

Aunque la idea era muy innovadora, su éxito no fue tan grande. En 2001, la compañía cerró debido a la creación de nuevos competidores que le comieron todo el terreno en el mercado de las redes sociales.

Ilustración 15 - Sixdegrees, la primera red social

1999

La curiosidad de un programador informático británico por saber cómo era la vida de sus antiguos compañeros de instituto le inspira para que en este año se desarrolle Friends Reunited.

A finales del año 2000, contaba con 3.000 miembros y tan sólo un año después, esta cifra alcanzaba el valor de 2.500.000.

En la actualidad Friends Reunited pertenece a la empresa DC Thompson y sigue siendo accesible para los usuarios, aunque con muy poco éxito.

2002

Este año se crea una nueva red social en Estados Unidos llamada Friendster que en sólo tres meses consigue alcanzar los tres millones de Usuarios. El servicio P2P ofrecido por Napster inspira a su creador para desarrollar un servicio en el que se pueda compartir algo más que archivos.

Para el año 2004, Friendster seguía siendo la red social más utilizada y con mayor número de usuarios hasta que MySpace la desbancó.

2003

En este año, dos de las redes sociales con más éxito de la historia son lanzadas.

Por un lado, aparece MySpace como una copia de las características de Friendster. En un primer momento, tan sólo los empleados de eUniverse, la empresa desarrolladora, podían acceder a sus servicios. Meses más tarde se lanzaría al público en general apoyada por la gran capacidad de marketing de la compañía, consiguiendo desbancar a Friendster en poco tiempo. MySpace permitiría a sus usuarios modificar el código HTML de sus páginas a través de la creación de listas y la inclusión de música en sus perfiles.

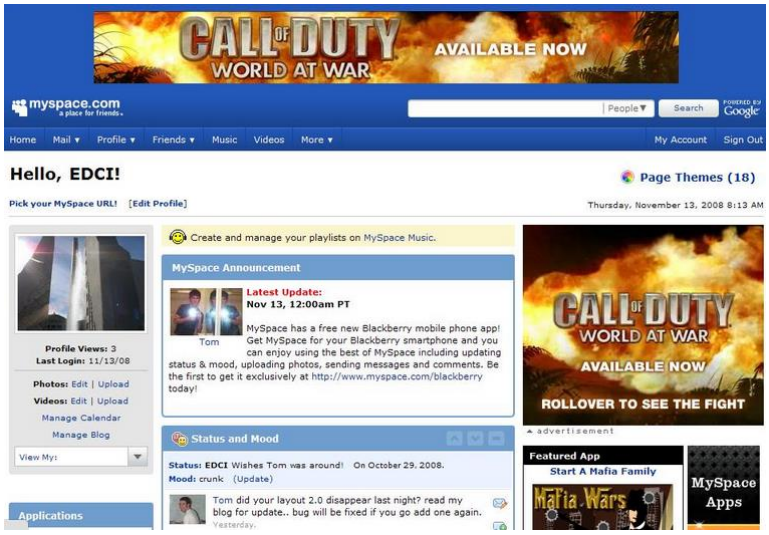


Ilustración 16 - Perfil de MySpace

En el año 2011, después de varios cambios de manos, MySpace tenía 33 millones de usuarios en sus redes.

Por otro lado, una red social orientada a los negocios es lanzada en 2003. LinkedIn ofrecía a sus usuarios un espacio donde mantener un perfil profesional donde contener a sus contactos, su currículum y sus más destacables acciones empresariales.

2004

En febrero de este año, estudiantes de la universidad de Hardvard lanzan a la web un nuevo servicio dirigido a sus compañeros de campus llamado Facebook. En un comienzo, este servicio estaba limitado a sus compañeros de universidad, aunque rápidamente se comenzó a expandir por otros campus en el área de Boston.

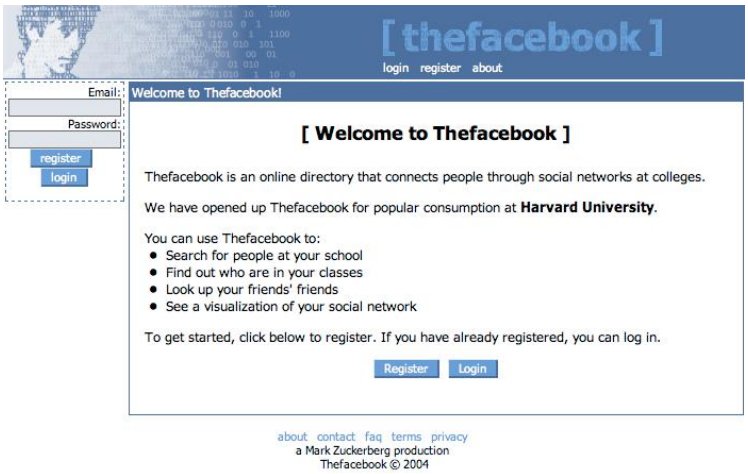


Ilustración 17 - Página de entrada de thefacebook, predecesor de Facebook

Gradualmente, este servicio se fue haciendo popular entre todos los universitarios de Estados Unidos y Canadá hasta alcanzar los cien millones de usuarios en agosto de 2008, periodo durante el cual el número de usuarios había crecido a un ritmo del 178.83% mensual.

A día de hoy, Facebook es la red social con mayor número de usuarios y mayor uso con 1.230 millones de usuarios a finales del año 2013.

Facebook es una de las compañías tecnológicas mejor valoradas en bolsa y recientemente ha protagonizado grandes titulares con la compra de Whatsapp 19.000 millones de dólares.

2006

En España nace en este año una nueva red social basada en la funcionalidad de Facebook llamada Tuenti. En un primer momento el acceso era restringido a usuarios y más tarde, por invitación. En el año 2010 es adquirida por Telefónica y un año más tarde se elimina la restricción de la invitación permitiendo el acceso libre a usuarios.

En la actualidad Tuenti cuenta con más de 15 millones de perfiles de usuarios, lo que supone prácticamente un tercio de la población de España.

En este mismo año, un nuevo concepto de red social es lanzado en Estados Unidos. Twitter es una red social con características de microblog, donde se permiten las entradas de hasta 140 caracteres.

Twitter experimentó un rápido crecimiento desde sus inicios alcanzando 1.600.000 tweets publicados a lo largo del año en el 2007. En septiembre de 2013, un estudio de la compañía muestra que 200 millones de usuarios mandan 400 millones de tweets diariamente. Twitter se ha convertido en una herramienta de trabajo para el mundo de las comunicaciones y constituye la primera puerta de acceso de los usuarios hacia miles de deportistas, políticos y periodistas.

En la actualidad, cada vez son más empresas las que están apostando por la creación de RRSS internas para que sus empleados puedan compartir información y acceder a los datos de los clientes de una manera más rápida con acceso desde cualquier dispositivo. Esta evolución tiene su lógica: al igual que los chats y la mensajería instantánea abrieron el camino a la creación de RRSS para el público, la tendencia es que pase lo mismo en las empresas con un acceso privado.

En concreto, el uso de este tipo de herramientas en el ámbito hospitalario puede mejorar mucho dos aspectos claves de este trabajo: disminuir la pérdida de información en la transmisión entre los distintos turnos de trabajo y mejorar la asistencia telemátiva de pacientes.

2.5 APIs sociales

Las APIs sociales permiten realizar llamadas remotas hacia el servidor de la red social para acceder a contenidos de los perfiles de los usuarios como listas de contactos o información del contacto en sí.

2.5.1 APIs para construir redes sociales

Este tipo de APIs, además de permitirnos acceder a los datos de perfil del usuarios, nos permitirán construir una nueva red social, y de esta manera, poder asegurar la privacidad de los pacientes y la confidencialidad de sus datos.

Social API

[16] Social API facilita la integración a los navegadores web con los contenidos sociales de aquellas redes sociales que han sido implementadas por Firefox.

Social API es un conjunto de herramientas que permiten por tanto obtener datos de las redes sociales que han sido desarrolladas por Firefox y que permiten la autenticación desde cualquier sitio web hacia esta red social.

Esta herramienta aún está en desarrollo y la documentación se encuentra incompleta.

OpenSocial

[9, 10] Se puede definir OpenSocial como un conjunto de especificaciones APIs que permiten construir aplicaciones sociales para distintos sitios webs. Este conjunto de APIs están escritas para ser utilizadas con *javascript* y con Google Data.

OpenSocial fue lanzado en 2007 por primera vez. Esta primera versión recibió muchas críticas, ya que sólo funcionaba correctamente en Orkut (plataforma contenedora creada por Google), y no con todos los Gadgets. De hecho, esta primera versión de OpenSocial fue crackeada en tan solo veinte minutos. A partir de entonces, diferentes versiones fueron lanzadas, permitiendo por un lado ser soportadas en diferentes sitios web y mejorando las prestaciones de seguridad.

Son varias las ventajas de OpenSocial para la creación de este tipo de aplicaciones, pero por encima de todas podemos destacar, que cualquier sitio web puede contar con aplicaciones OpenSocial, y por lo tanto los desarrolladores solo deberían de aprender a manejar un API para crear este tipo de aplicaciones. Esto da una idea de las posibilidades que abre este proyecto para el mundo de las aplicaciones sociales, ya que los desarrolladores poseen ahora muchos más canales de distribución para llegar a los usuarios.

Se dice que OpenSocial fue lanzado por Google buscando un movimiento de los desarrolladores de Facebook hacia su herramienta, aunque realmente aún no es muy popular entre el usuario medio, hay muchas empresas que están apostando por OpenSocial para desarrollar sus aplicaciones sociales, como IBM, eXo Platform o Jive Software.

2.5.2 APIs para conectar y hacer uso de otras redes sociales

Este tipo de APIs tan sólo nos permitirán acceder a los datos de los perfiles de los usuarios de RRSS públicas. Algunas de ellas nos permitirán acceder a los datos de varias RRSS, y otras serán específicas para una red social.

HybridAuth API

[17] HybridAuth permite a los desarrolladores implementar aplicaciones sociales para que los clientes puedan disfrutar de los servicios de autenticación, compartición de archivos, listas de amigos, etc. y que actúa como enlace de la aplicación y varios proveedores de identidad como Facebook, Youtube o Twitter.

Básicamente, HybridAuth nos permite realizar dos tipos de acciones:

1) **Autenticación del usuario:** El usuario es redirigido a la página de su proveedor de identidad para que introduzca su usuario y contraseña. Cuando el usuario vuelve a la página de la que venía, el proveedor de identidad envía un token que indica si la autenticación ha sido exitosa o fallida.
2) **Autorización de acceso:** Una vez el usuario se ha autenticado, HybridAuth pide acceso al proveedor de identidad para acceder a datos de tu perfil, amigos, listas de contactos, etc.

Características de HybridAuth

- Permite la autenticación a través de varios de las más conocidas redes sociales de una manera sencilla
- Permite obtener el perfil de una manera estandarizada para todas las redes sociales
- Permite obtener datos del perfil como listas de amigos y establecer otros como el estado del usuario.
- Simplifica el proceso para obtener un proveedor de identidad para tu sitio web
- Todas las acciones son transparentes para el usuario final

Usos de HybridAuth

Este API puede ser usado para muchas acciones generales como la de activar la autenticación de usuarios mediante servicios populares como Facebook o Twitter; permitir a los usuarios compartir los comentarios de tu aplicación en estas redes sociales; desarrollar aplicaciones sociales para incluirlas en blogs, webs, etc.

Google+ API

Este API es el interfaz de programación para los gadgets de Google +. Se suele usar para integrar la aplicación o sitio web con Google+, lo que permite una mayor participación de los usuarios usando algunas de las funciones de Google.

De esta manera, el API nos permitirá acceder a los datos del perfil de Google+ desde una aplicación.

Para acceder al perfil, este API utiliza los servicios de OAuth 2.0 (servicio similar a OpenID) para que el usuario permita a la aplicación el acceso a sus datos. Una vez esté activado el permiso, el sistema se basa en llamadas HTTP de tipo GET hacia el API para pedir los recursos.

Para solicitudes que puedan ser respondidas con colecciones de elementos largas, se puede habilitar un número máximo de elementos (por defecto son 20 elementos).

OneAll

OneAll es una empresa tecnológica que ofrece un conjunto de herramientas web y un API unificado para todos los medios sociales para facilitar la integración de las redes sociales en los sitios web como son Facebook, Twitter, Google, Yahoo y Linkedin en las empresas y en las páginas personales.

El API de OneAll unifica más de 25 redes sociales y proporciona una solución única para la interacción con todas ellas. Permite trabajar con múltiples redes sociales obteniendo elementos estructurados de manera similar sin importar la red social con la que se trabaja.

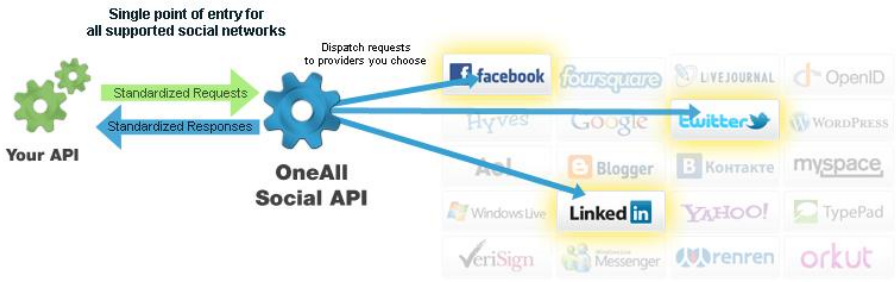


Ilustración 18 - Diagrama de interacción del API OneAll

Características

- **Facilidad para la integración de redes sociales:** A través de la instalación de plugin Turnkey y del API JSON/REST se puede integrar fácilmente los servicios de más de 25 redes sociales en tu sitio web.
- **Autenticación social:** El servicio que se ofrece de *Social Login Service* permite autenticarse con un solo click para permitir el acceso a los datos sociales.
- **Comparte con tus amigos:** Permite acceso al servicio *Social Sharing Service* para que los usuarios puedan compartir comentarios, compras o actividades en tu sitio web con sus amigos de muchas redes sociales.
- **Social insight:** Mediante este servicio se puede realizar un seguimiento del impacto que tiene la implantación de OneAll para tu empresa.

2.5.3 Conclusiones

Para la realización de esta elección partiremos de una premisa básica: dado que se trata de una aplicación para un entorno hospitalario se busca que el API nos proporcione una independencia de servicios externos ofrecidos por otras empresas. Esta sería la única manera de asegurar la confidencialidad de los datos de los pacientes.

Por ello, quedan descartados aquellos APIs que no permitan construir una red social.

La discusión se centraría entonces entre Social API y OpenSocial. Teniendo en cuenta que Social API está aún por desarrollar, que la documentación es escasa y desactualizada, nos decantaremos finalmente por OpenSocial y en concreto, utilizaremos un servidor Apache Rave como contenedor para la aplicación.

Esta decisión nos permitirá controlar por completo los usuarios de la red social para poder administrarlos.

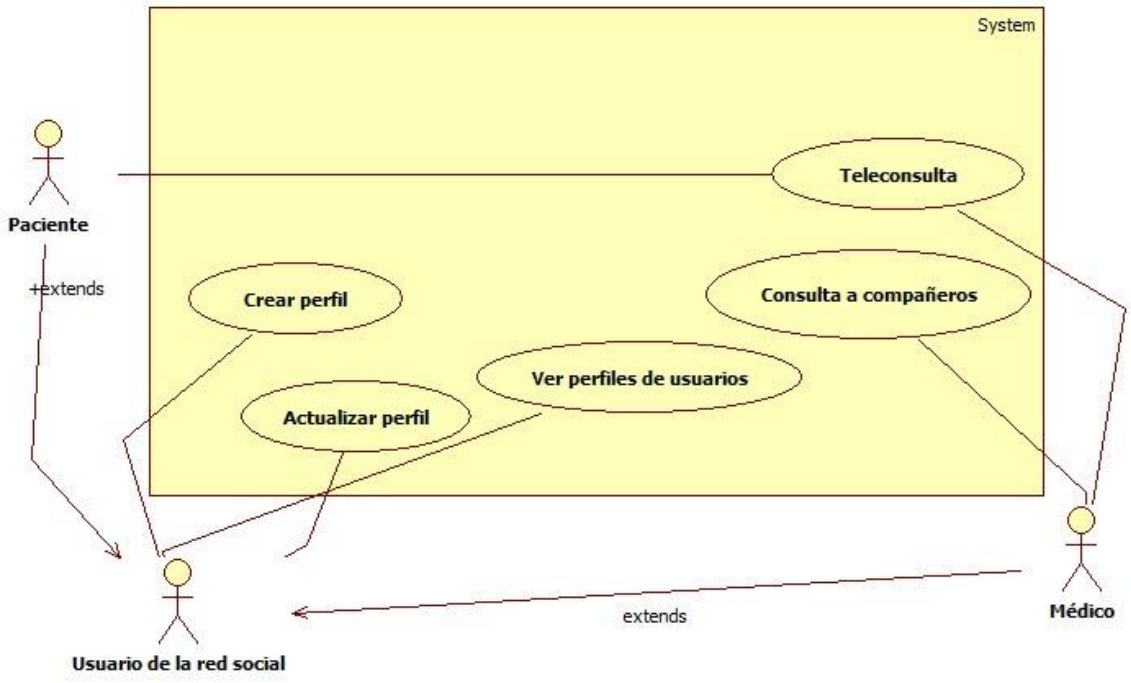
3. Desarrollo

El desarrollo del proyecto se dividió en tres fases principales: inicial, desarrollo e integración. En este capítulo especificaremos los cambios implementados durante estas fases y además detallaremos la arquitectura utilizada, así como los requisitos y casos de uso.

3.1 Casos de uso

En este apartado se pretenden describir los casos de uso que explican el comportamiento de la aplicación ante los eventos generados por los usuarios.

3.1.1 Diagrama de casos de uso



3.1.2 Casos de uso

Título	Consulta a compañeros
Descripción	Los componentes de un mismo equipo médico pueden compartir información sobre los distintos pacientes que estén a su cargo. Para que puedan compartir la información, los usuarios implicados deben de aceptar previamente en sus listas de usuarios a los otros implicados.
Objetivo	Permitir intercambio de información
Actores	Médicos

Tabla 3 - Caso de uso 1: Comunicación entre equipo médico

Título	Teleconsulta
Descripción	Los pacientes tendrán acceso a la aplicación y de esta manera podrían consultar a los médicos de manera telemática. Para que esta acción fuera posible, el paciente debería ser aceptado previamente por el médico pertinente en su lista de contactos.
Objetivo	Mejorar la asistencia al paciente
Actores	Pacientes y médicos

Tabla 4 - Caso de uso 2: Teleconsulta

Título	Crear perfil
Descripción	Los usuarios de la red social podrán crearse un perfil desde la página de acceso a la red social que generará un nuevo usuario en el sistema
Objetivo	Gestión de usuarios
Actores	Usuarios de la red social

Tabla 5 – Caso de uso 3: Crear perfil

Título	Actualizar perfil
Descripción	Los usuarios de la red social podrán actualizar su perfil desde la página de la red social.
Objetivo	Gestión de usuarios
Actores	Usuarios de la red social

Tabla 6 – Caso de uso 4: Actualizar perfil

Título	Ver perfiles de usuarios
Descripción	Los usuarios de la red social podrán acceder a la información del perfil de otros usuarios
Objetivo	Gestión de usuarios
Actores	Usuarios de la red social

Tabla 7 – Caso de uso 5: Ver perfiles de usuarios

3.2 Requisitos

Los requisitos han ido cambiando durante las distintas fases de desarrollo del proyecto. En cada etapa se han revisado los mismos para probar su cumplimiento y para revisar si era necesario realizar algún cambio en la lista.

ID	Requisito	Descripción
REQ_1	El sistema debe de estar formado por elementos de código abierto	Todos los elementos que conformen el sistema para su correcto funcionamiento han de ser elementos de código abierto para facilitar la implementación del sistema.
REQ_2	El sistema ha de permitir mensajes de gestión	El cliente debe de poder gestionar su status dentro de la aplicación y se deben permitir mensajes entre los sub sistemas que integran la aplicación
REQ_3	El sistema debe de tener una funcionalidad de red social	Las funciones que el sistema debe de facilitar son: chat one-to-one, presencia, actualización de rosters, actualización del estado del cliente
REQ_4	La red social debe de ser capaz de servir otros futuros gadgets	El API Social que se utilice debe de permitir la creación de una red social que a la vez actúe como contenedor para el gadget que se va a implementar y para otros gadgets que puedan completar la aplicación en el futuro
REQ_5	La Interfaz gráfica debe ser simple y funcional	Los usuarios de la red social deben de encontrar el gadget implementado sencillo de utilizar, de manera que su aspecto sea intuitivo y no sea necesario un manual para utilizarlo
REQ_6	El sistema debe de tener una gestión única de usuarios	Aunque el sistema en su conjunto esté integrado por varios sub-sistemas, debe de ser uno de ellos quién mantenga una autoridad sobre los otros y gestione los usuarios y sus perfiles

Tabla 8 - Requisitos de la aplicación

Por aclaración, nos referimos como gestión única de los usuarios a la creación y mantenimiento de usuarios, credenciales y datos de manera única para toda la aplicación.

3.3 Diseño

En este capítulo de la memoria, abordaremos el proceso de diseño de la aplicación, especificando la arquitectura escogida para su correcto funcionamiento, así como los componentes de la misma.

3.3.1 Arquitectura

Necesitamos de una integración total de las arquitecturas de ambos sub-sistemas, de tal manera que el usuario tenga una experiencia integral sin necesidad de conocer que hay detrás del gadget que está utilizando. Barajamos dos aproximaciones para una única solución: primero buscamos unificar las bases de datos de ambos servidores para que sólo haya una que maneje la gestión de los usuarios, o relacionando ambas para que sea una de las dos una base de datos maestra sobre la otra; por otro lado, tratamos de buscar una solución software en tiempo real, a través del plugin *User Service* para Openfire, que nos permite mandar peticiones a la consola de administración del servidor Openfire para gestión de usuarios, de tal manera que se pueden añadir o borrar usuarios y gestionar suscripciones sin necesidad de utilizar la consola del servidor XMPP.

Las dos alternativas quedan reflejadas en el siguiente gráfico:

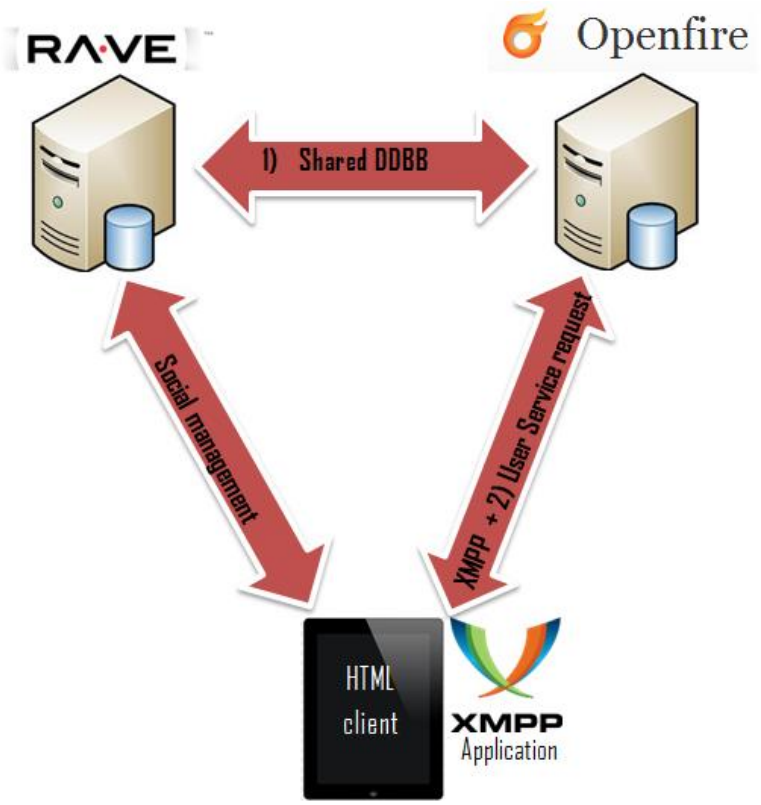


Ilustración 19 - Estructura de la aplicación en la fase de integración compartiendo BBDD

Descartaremos la opción de compartir las bases de datos por una razón de complejidad en la instalación de los plugin necesarios para gestionar ambos servidores desde bases de datos externas y apostaremos por el uso el plugin *User Service* de Openfire por sencillez en la instalación y facilidad de gestión

Para que la autenticación sea única, debemos de obtener los datos de autenticación de Apache Rave una vez el usuario está dentro de la red social. Evidentemente, tan sólo podremos obtener el dato del nombre del usuario y nunca la contraseña, pues debe de estar protegida. A través del API OpenSocial, obtenemos el dato del nombre de usuario lanzando una petición desde el cliente al servidor Apache Rave. Una vez obtengamos este dato, mandamos una petición HTTP a la consola de administración del servidor Openfire, gracias al plugin *User Service*, para que cree un nuevo usuario con ese nombre y una contraseña fácilmente gestionable. En caso de que el usuario ya haya sido creado anteriormente, el servidor se limitará a rechazar la petición. En caso contrario, creará el usuario con las credenciales solicitadas en la petición HTTP mandada desde el cliente. Es importante remarcar que para que las peticiones sean aceptadas por el servidor Openfire,

deben de incluir una clave secreta tan solo gestionable desde la consola de administración del servidor.

Estando ya creado el nuevo usuario en el servidor XMPP (o antiguo, en caso de existir previamente en el servidor), procedemos a la conexión automática mediante las credenciales que ya conocemos. De esta manera, el gadget se conecta automáticamente con el mismo usuario que tenemos en la red social, sin necesidad de introducir dos veces las credenciales.

A través del proceso explicado anteriormente, conseguimos que la gestión de usuarios sea única, ya que en todo momento, en el servidor XMPP, tan solo existen los usuarios que previamente existían en el servidor Apache Rave.

Finalmente, para que también la gestión de los amigos sea única también, una vez accedamos a la red social, a través de una petición al servidor Apache Rave, capturamos los amigos del usuario en la red social. De la misma manera que anteriormente, mandamos una petición HTTP al servidor Openfire añadiendo como amigos del usuario una vez estemos conectados al servidor. De nuevo necesitaremos tener el plugin *User Service* instalado en nuestro servidor Openfire para que estas peticiones sean atendidas por el servidor, y de la misma manera necesitamos incluir en las peticiones la clave secreta compartida entre los usuarios de la aplicación XMPP y el mismo servidor.

Por lo tanto, una vez valoradas las opciones barajadas, en este punto la arquitectura final de nuestro proyecto queda definida en la siguiente imagen.

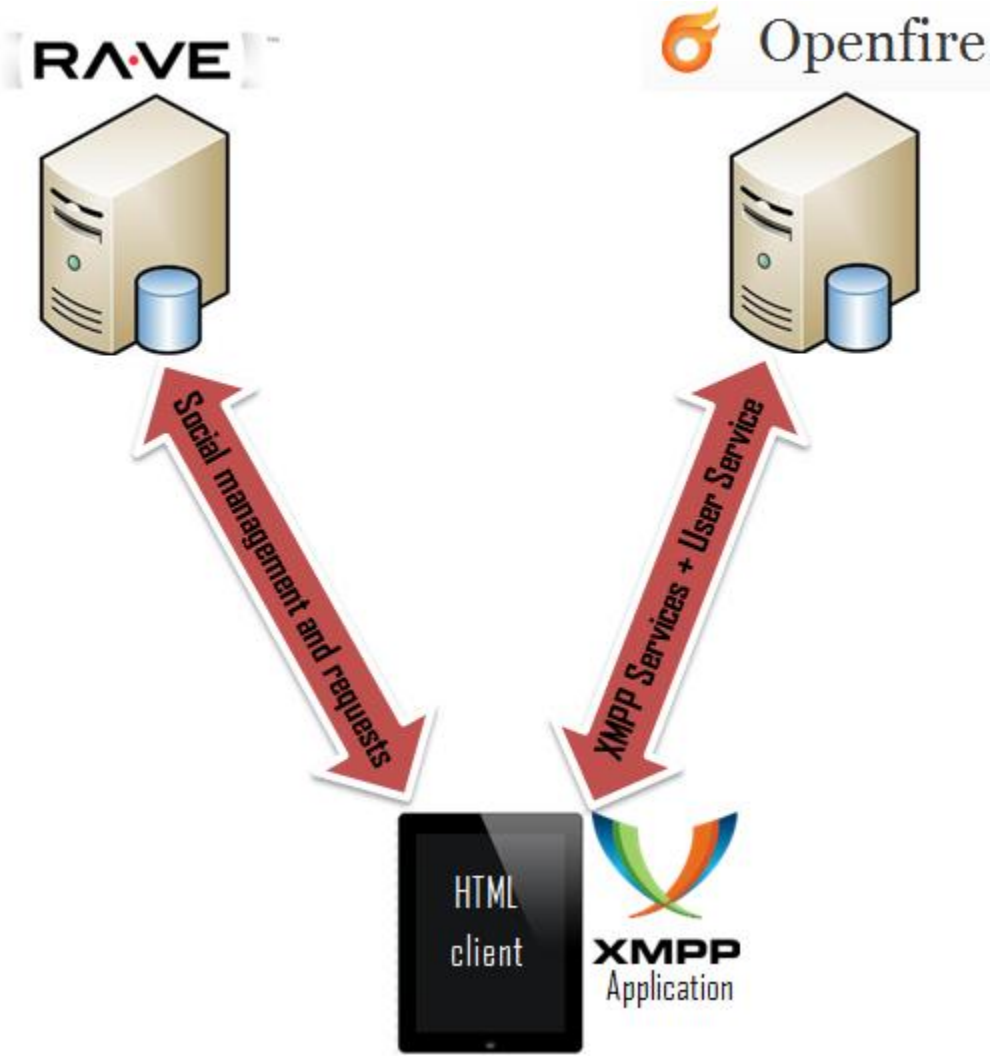


Ilustración 20 - Estructura de la aplicación con User Service

Tal y como se puede ver en la imagen, tan sólo existe tráfico entre el cliente y ambos servidores, y la gestión única de usuarios se realiza finalmente desde el propio cliente a través de peticiones a ambos servidores.

Comunicación cliente-Openfire

Es evidente que la principal comunicación que habrá entre un cliente XMPP y el servidor Openfire serán, precisamente, stanzas XMPP. Recordemos que este tipo de stanzas son 3 principales: la correspondiente a los mensajes *<messages/>*; la correspondiente a las request *<iq/>*, donde se incluyen también peticiones de roster o

actualización del mismo; y finalmente las correspondientes a la información de presencia *<presence/>*.

Sin embargo, podría haber otro tipo de comunicación vital para nuestra aplicación: información sobre nuevos usuarios de la red. Esto se refiere a que desde la propia aplicación se intentará implementar una herramienta para la creación de nuevos usuarios sin necesidad de crearlos desde la consola de administración de Openfire.

Para ello, en el desarrollo de la aplicación, deberíamos de intentar que la información de presencia de los amigos del usuario esté presente durante toda la ejecución, incluso cuando pueda haber pérdidas de cobertura, desconexiones o reconexiones a la aplicación. Este hecho, aunque parezca trivial, no lo será tanto teniendo en cuenta las características de XMPP (sólo se usan mensajes PUSH desde el cliente), por lo que deberá ser el servidor quién gestione y quién resuelva la situación de un usuario desconectado repentinamente.

Comunicación Cliente-Rave

En este caso, la comunicación entre el cliente XMPP y el servidor Apache Rave tendrá menos peso en el desarrollo de la aplicación, pero sin embargo, quizás la información más sensible pase por este canal.

La idea del proyecto es que el cliente sepa en todo momento cuales son los amigos (el roster de la red social), para así comprobar cuáles de ellos disponen del widget instalado en su portal, y así aparezcan como amigos en la red XMPP. De alguna manera, sería necesario relacionar ambos servidores (al menos en la parte de usuarios amigos, que es la que nos interesa). Aunque este hecho, realmente nos plantea la posibilidad de que ambos servidores estén relacionados directamente entre ellos.

Gestión de usuarios vía User-Service

User Service es un plugin que se instala en el servidor Openfire y que nos permite lanzar peticiones HTTP a la consola de administración de éste para realizar cambios en las listas o agregar nuevos usuarios.

User service nos será útil para evitar la duplicidad de listas de contactos entre ambos servidores y lo utilizaremos en dos casos:

En primer lugar lo utilizaremos al comenzar la ejecución de la aplicación. Se extraerán los datos de usuarios del perfil a través del API OpenSocial y se utilizarán estos datos para crear un nuevo usuario en la base de datos de Openfire, mediante una petición HTTP a la consola de administración de este. Si el usuario está creado anteriormente, el servidor omitirá la petición.

En segundo lugar, se utilizará para que las listas de contactos sean la misma en ambos servidores. Se extraerá esta lista a través del API OpenSocial y se lanzará una petición hacia el servidor Openfire por cada elemento de la lista. De igual manera, si el usuario ya estaba en la lista, se omitirá la petición.

3.3.2 Componentes de la aplicación

En este sub-apartado, trataremos de estudiar cuáles son los elementos que van a conformar el diseño de la aplicación y las relaciones que existirán entre todos estos elementos.

Resumido de una manera breve: la aplicación permite intercambiar mensajes XMPP, está implementada en código *javascript*, apoyándose en la librería Strophe, que permite que cualquier aplicación XMPP funcione en cualquier navegador a través de su API para este lenguaje de programación.

Introducción

El contenedor para este gadget, que vamos a utilizar como red social como ya hemos comentado antes, será **Apache Rave**, que a la vez servirá todos los contenidos de la red social y administrará sus usuarios. Existen otros contenedores que se podrían utilizar como el entorno para desarrolladores de gadgets ideado por Google, Orkut.

Para la implementación del gadget utilizaremos el protocolo de comunicación **XMPP**, que nos permitirá intercambiar mensajes entre los usuarios de la red XMPP a través de un servidor, y que es muy utilizado para aplicaciones empresariales como *Webex*

de Cisco e incluso también en las aplicaciones más utilizadas en los dispositivos móviles como *Whatsapp* o *Telegram*.

Para que el trabajo de implementación de la aplicación no sea tan costoso, utilizaremos una librería *javascript* para XMPP llamada **Strophe**, que por un lado nos permitirá programar con mayor facilidad a un nivel más alto, y por otro lado nos permitirá ensamblar toda la aplicación en un documento XML, que es el formato de los gadgets OpenSocial.

Para el manejo de los elementos HTML de la aplicación y la interfaz gráfica utilizaremos las librerías **jQuery** y **jQuery UI**, que aportarán dinamismo en la aplicación y simplificación al código.

Finalmente, como ya hemos comentado, utilizaremos un servidor XMPP para servir a los clientes de la aplicación. Se tratará de un servidor **Openfire**. Este servidor es de fácil instalación y dispone de su propia consola de administración, desde la que poder gestionar los usuarios, la base de datos utilizada, opciones sobre los puertos o desconexiones de usuarios.

XMPP

Historia de XMPP

[3] En enero de 1999, Jeremie Miller anuncia la existencia de Jabber, una tecnología opensource para mensajería instantánea. A partir de un primer servidor abierto (jabbered) se comienzan a implementar varios clientes y librerías de código.

En mayo del 2000, una primera versión 1.0 del servidor jabbered es lanzada y con ella el paquete de características soportadas (streaming XML, mensajería instantánea, presencia, listas de contactos, etc.)

Se forma la Jabber Software Foundation (JSF) en agosto del 2001 con el propósito de coordinar los protocolos usados en la comunidad Jabber/XMPP. Se comienzan a documentar estos protocolos y se comienzan a desarrollar algunas extensiones.

A finales de octubre de 2002, se forma oficialmente el grupo XMPP Working Group, una vez aprobado por la IESG (Internet Engineering Steering Group). De esta manera la IETF (Internet Engineering Task Force), encargada de la estandarización de los protocolos de Internet, acepta encargarse del nuevo protocolo XMPP. Se comienza a trabajar en una versión estandarizada del XMPP Core.

En octubre de 2004 se publican los primeros estándares RFC de XMPP. Son cuatro: *RFC 3920: Extensible Messaging and Presence Protocol (XMPP): Core; RFC 3921: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence; RFC 3922: Mapping the Extensible Messaging and Presence Protocol (XMPP) to Common Presence and Instant Messaging (CPIM); RFC 3923: End-to-End Signing and Object Encryption for the Extensible Messaging and Presence Protocol (XMPP)*. Comienza así la expansión de este protocolo.

A partir de 2005, se crean más de una docena de servidores XMPP escritos en distintos lenguajes de programación. Y así, diversas compañías como Google con GTalk, se lanzan a implementar aplicaciones de mensajería instantánea usando XMPP.

En los siguientes años el protocolo se sigue desarrollando, los estándares se actualizan y se añaden diferentes extensiones que permiten, por ejemplo la implementación de chat multi-usuarios.

Arquitectura XMPP

[1,2] XMPP presenta una arquitectura muy similar a la propia World Wide Web o el email. Se trata de una arquitectura cliente-servidor descentralizada, en la que se puede soportar varios miles de servidores (de tipo Ejabbered, Openfire, etc.). La siguiente imagen, nos da una idea de la arquitectura de una red XMPP de tamaño medio en la que se interconectan varios servidores (comunicados también con el protocolo XMPP) para poder servir a un mayor número de usuarios.

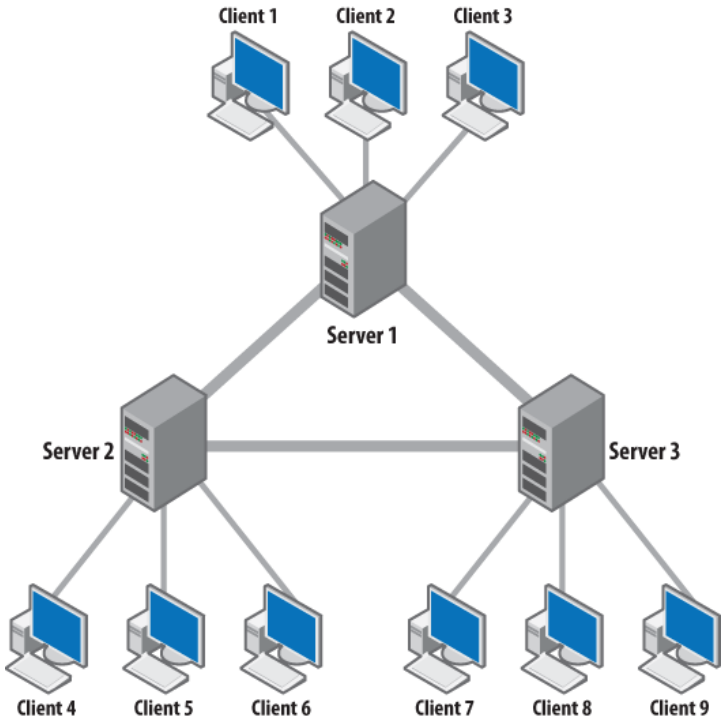


Ilustración 21- Arquitectura de redes XMPP

A pesar de las similitudes en la arquitectura, se encuentran diferencias importantes entre estas tres tecnologías.

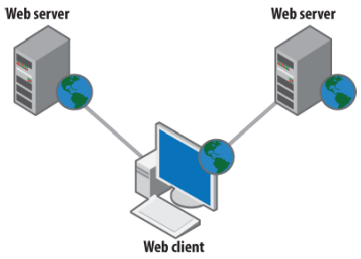
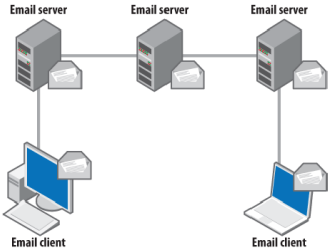
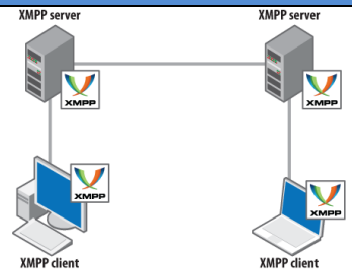
Web service	Email	XMPP
		
Las conexiones Web se producen entre cliente y servidor o varios servidores simultáneamente (no permite conexión servidor-servidor)	La red email utiliza las conexiones entre varios servidores hasta llegar al usuario final	La red XMPP sólo permite una conexión servidor-servidor en su arquitectura.

Tabla 9 - Tabla comparativa de arquitecturas para redes HTTP, email y XMPP (1)

La siguiente tabla puede resumir las diferencias entre estas tres arquitecturas:

Característica	Web	Email	Jabber
Conexiones entre dominios	No	Sí	Sí
Saltos múltiples	N/A	Sí	No

Tabla 10 - Tabla comparativa de arquitecturas para redes HTTP, email y XMPP (2)

Strophe

Strophe

[4]El término *Strophe* proviene originalmente del término poético con el mismo nombre que se refería a la primera parte de una oda en la Antigua tragedia Griega, es por eso que Strophe es definido por sus propios creadores como una librería para poetas XMPP.

En concreto, la librería que se va a utilizar en este proyecto será la librería Strophe.js, es decir, una librería *javascript* pensada para la implementación de clientes (exploradores en su gran mayoría). Usa **BOSH**, de lo cual hablaremos a continuación en este capítulo.

BOSH: Bidirectional-streams Over Synchronous HTTP

BOSH es un protocolo de la capa de transporte que emula un flujo de datos bidireccional entre dos entidades usando varios pares de conexiones síncronas HTTP del tipo pregunta-respuesta, sin la necesidad de utilizar polling. Por lo tanto, BOSH será el responsable de facilitar el uso de mensajes tipo "push" de nuestra aplicación.

Para que se realicen este tipo de operaciones "push" en nuestra red, un cliente BOSH ha de comenzar una petición HTTP, y el servidor esperará a responder a esta petición hasta que no tenga datos que enviar. Una vez tiene algo que enviar, lo envía al cliente, y éste responderá con otra petición al servidor, de tal manera que siempre que el servidor

tenga algo que enviar, lo podrá hacer si necesidad de que el cliente esté haciendo polling constantemente sobre el servidor.

En nuestro caso, el propio servidor Openfire será quien acepte las conexiones tipo BOSH, y el cliente que programemos, a través de la librería Strophe, quién genere la conexión. Si nuestro servidor no incluyera este tipo de soporte, deberíamos de utilizar un gestor de conexiones BOSH como pudiera ser *Punjab* o *JabberHTTPBind*.

¿Qué funciones de Strophe son útiles para nuestra aplicación?

Strophe.Connection = function (service, options): Esta función la utilizaremos para generar la conexión entre el cliente y servidor. Para ello (dado que el servidor lo permite sin necesidad de un gestor de conexión BOSH), ejecutaremos en la primera línea de código (una vez el documento este cargado) esta sentencia:

connection = new Strophe.Connection()

De esta manera, conseguimos que nuestro cliente comience una conexión BOSH con el servidor previamente configurado.

function $msg (attrs): Esta función es un constructor para un elemento de tipo *Message*, lo que permite generar una stanza de este mismo tipo. En el parámetro *attr*, se incluye una cadena de caracteres en la que se especifican los atributos de la stanza *Message* a enviar. Lo podemos ver en el siguiente ejemplo. A partir del método *.c()* adjuntamos a la stanza un nuevo elemento hijo.

$msg({to: docAllison@hospital, from: 'doctorMirza@hospital/office', type: 'chat'}).c('body').t(Let's take a break!)

De esta manera generamos una stanza *</message>* especificando los atributos y los hijos de la misma.

function $pres (attrs): Al igual que la función anterior, ésta genera un elemento tipo presencia que genera una stanza *</pressence>.* También en el parámetro *attrs* se incluye una cadena de caracteres especificando los atributos de dicha stanza.

$pres().c('show').t('away')

function $iq(attrs): De nuevo, esta función genera un elemento que permite crear una stanza, en este caso de tipo </iq>. Al igual que en los casos anteriores el parámetro *attrs* permite especificar los atributos de esta stanza.

$iq({type: "set"}).c("query", {xmlns: "jabber:iq:roster"}).c("item", {jid: 'docBird@hospital', name: 'Dr.Bird'})

En esta IQ stanza estamos añadiendo al contacto con JID *docBird@hospital* a nuestro roster.

send: function (elem): Esta función sirve para enviar cualquier tipo de stanza. Se empujan los datos XML hacia la cola de envío. Cuando se envía cualquier petición BOSH al servidor, toda la cola se vacía.

sendIQ: function(elem, callback, errback, timeout): Esta función sirve para enviar tan solo stanzas IQ. La diferencia con la función anterior, radica en las posibilidades que se puede incluir en este envío a través de los parámetros de la función. El parámetro *callback* permite especificar la función a llamar ante la respuesta a esta stanza; *errback* permite llamar a una función específica ante una respuesta de tipo <error> ante nuestra stanza; Finalmente *timeout*, añade la posibilidad de incluir un timer (en milisegundos) ante este envío.

addHandler: function (handler, ns, name, type, id, from, options): Quizás estemos ante la función clave de esta librería, ya que nos permite añadir handlers para reaccionar ante eventos que pasen, como por ejemplo, que algún usuario se quiera suscribir a nuestra lista, o llegue nueva información de presencia de otros usuarios. El parámetro *handler* permite especificar la función a la que llamar cuando se produce dicho evento; *ns* define una manera de especificar ante qué tipo de evento reaccionar buscando una cadena de caracteres que coincida dentro de una stanza recibida; *name* define el nombre de la stanza ante la que reaccionar; *type* especifica el tipo de stanza ante la que se lanza la función; el parámetro *id* establece la búsqueda del atributo id de la stanza; *from* especifica el emisor de la stanza; y finalmente *options* son opciones que se puede añadir al handler. Podemos ver una llamada a dicha función en el siguiente ejemplo:

connection.addHandler(onSubscription, null, "presence", "subscribe");

Se puede ver como se añade un nuevo handler que llama a la función *onSubscription* ante una stanza de tipo presence que busca suscribirse a nuestra lista.

Es importante, que para que un handler se conserve, la función a la que llama debe devolver true (como resultado del handler) para que este quede activo. De manera contraria, el handler desaparecerá y quedará borrado de la ejecución.

Gracias a todas estas funciones y los servicios que ofrece la librería Strophe podemos crear una aplicación XMPP programando en *javascript*. Un código ligero que fácilmente puede ser ejecutado en cualquier navegador, de cualquier dispositivo.

jQuery

[6] jQuery es una biblioteca *javascript* muy usada para interactuar con documentos HTML de una manera más sencilla. Permite manipular los elementos del árbol DOM (el árbol compuesto por cada uno de los elementos de una página web) con mayor facilidad.

La mayor utilidad de esta biblioteca es la de permitir cambiar el contenido de la página dinámicamente sin necesidad de recargarla, mediante la manipulación del árbol DOM y peticiones AJAX. Para ello se utiliza la función *$()* o *jQuery()*.

Función jQuery

La manera más sencilla de acceder a un elemento del árbol DOM es mediante el uso de esta función. La sintaxis es la siguiente:

$(#id) o jQuery(#id): donde *id* es el identificador del elemento DOM al que se quiere acceder

Una vez hemos accedido al elemento podemos modificarlo, añadiendo o eliminando código HTML del mismo, como muestra el siguiente ejemplo.

*$(#roster ul).append("
New user: Dr.Bird has arrived!
")*

De esta manera añadimos a la lista dentro del elemento DOM *roster* un nuevo elemento con el contenido especificado.

Estado del documento desplegado en el navegador

Esta librería también nos permite realizar acciones según el estado del documento. Por ejemplo, una vez se ha inicializado el documento:

```
$(document).ready(function() {
//Acciones a realizar una vez el documento está inicializado.
});
```

Pero también nos permite realizar acciones cuando se lanza algún evento dentro del documento:

```
$(document).bind('connect', function(ev, data) {
 //Acciones a realizar cuando se lanza el disparador de conexión
});
```

O cuando se genera una acción sobre un botón determinado:

```
$('#sendbutton').on('click', function() {
        //Acciones a realizar cuando el botón es pulsado
        });
```

jQuery UI

[7] Al igual que jQuery, jQuery UI también es una biblioteca *javascript* que añade a esta primera un conjunto de widgets y efectos visuales para la creación de aplicaciones web. Esta librería se puede personalizar directamente antes de descargarla.

No vamos a detenernos en toda la librería. Simplemente remarcar el uso que le daremos en nuestra aplicación: generaremos un sistema de pestañas a partir de esta librería mediante la función *tabs().*

Esta función nos permite generar un sistema de pestañas con unas características predeterminadas por la librería, añadir nuevas pestañas, movimiento entre ellas, borrar pestañas en desuso, etc. Por lo tanto, nos será muy útil para generar una interfaz gráfica en la que mostrar las conversaciones con distintos usuarios.

Openfire server

[8] Openfire es un servidor opensource con licencia Apache que adopta el protocolo XMPP. El servidor está escrito en Java y es muy usado para las aplicaciones de Mensajería Instantánea y Chat de Grupo.

Este servidor, se convertirá en parte fundamental de la aplicación objeto del estudio, ya que será él quien gestionará la conexión BOSH con el cliente y a través de él pasarán todos los mensajes entre los distintos usuarios.

Configuración Openfire con BOSH

[5] Configuraremos Openfire desde una instalación en local que haremos (podemos encontrar en la página web oficial varias distribuciones para distintos Sistemas Operativos), ya que es así donde en probaremos nuestra aplicación para después desplegarla en un servidor de la Universidad.

Para acceder a la consola, por tanto accederemos a la URL *http://localhost:9090*. La primera vez que accedamos, un asistente de configuración será lanzado en el servidor.

En la primera página configuraremos el idioma y le daremos a continuar. A continuación configuraremos los puertos y el dominio para la administración del servidor.

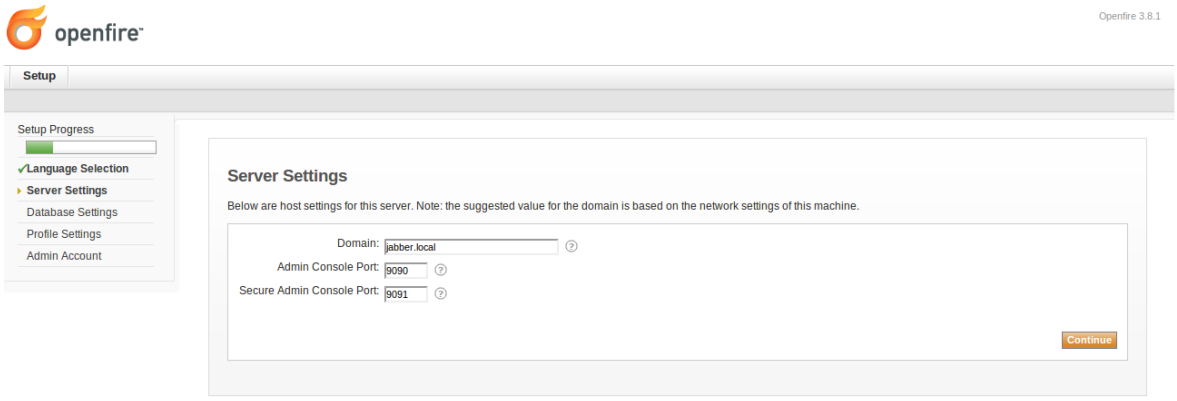


Ilustración 20 - Puertos de la consola Openfire

Como se puede ver, utilizaremos los puertos 9090 y 9091. El dominio es escogido por el administrador del servidor. Deberá ser reconocible, ya que será el mismo nombre que reciba el dominio de la red XMPP.

Pinchamos en continuar y aparece una nueva pantalla para configurar la base de datos. Podremos escoger entre una base de datos *H2*, o una base de datos externa. Más adelante hablaremos de las ventajas e inconvenientes que podemos encontrar utilizando una u otra.

En la siguiente ventana, tendremos la posibilidad de escoger en qué lugar queremos que se guarde la información de los perfiles de los usuarios: por simplicidad de las aplicaciones, lo normal es que escojamos la opción por defecto, aunque en un futuro se pueda estudiar otras opciones para plantear un escenario con un mayor número de usuarios.

Finalmente, en la última pantalla estableceremos las credenciales de acceso como administrador del servidor.

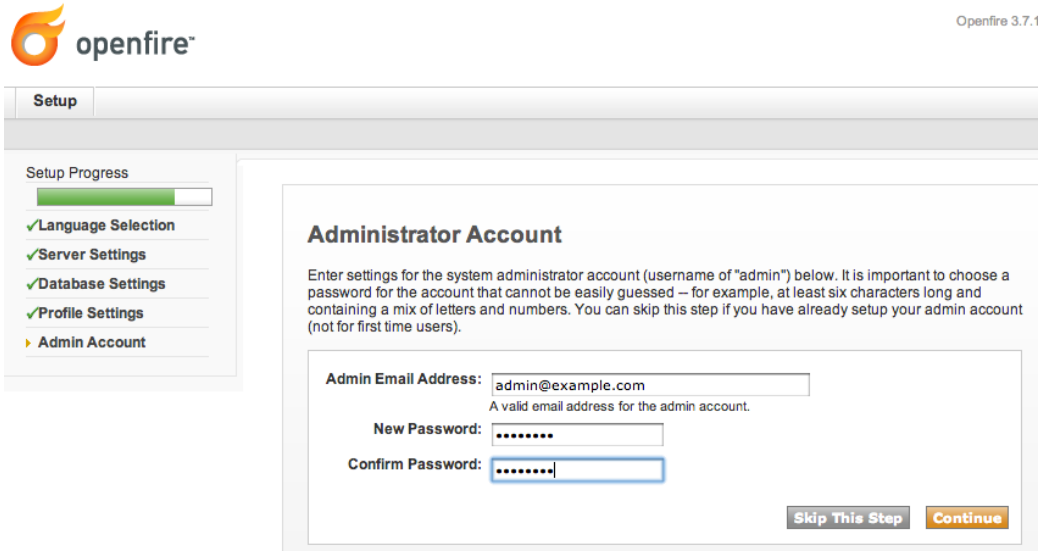


Ilustración 21 - Establecimiento de credenciales para el administrador en Openfire

Reiniciaremos el servidor para que surja efecto la nueva configuración y volveremos a entrar a la interfaz de administración. Necesitaremos de la contraseña anteriormente establecida. El usuario por defecto será “admin”.

El siguiente paso consistirá en configurar BOSH para que el servidor habilite este tipo de conexiones por parte de los clientes. Como ya hemos hablado en varias ocasiones, lo que funcionalmente permite BOSH es el intercambio de stanzas entre clientes XMPP y un servidor Openfire a base de operaciones "push". Para acceder a las opciones del servidor sobre BOSH vamos a Server Settings->HTTP Binding. El apartado sobre HTTP-Bind debe estar configurado con los siguientes parámetros:

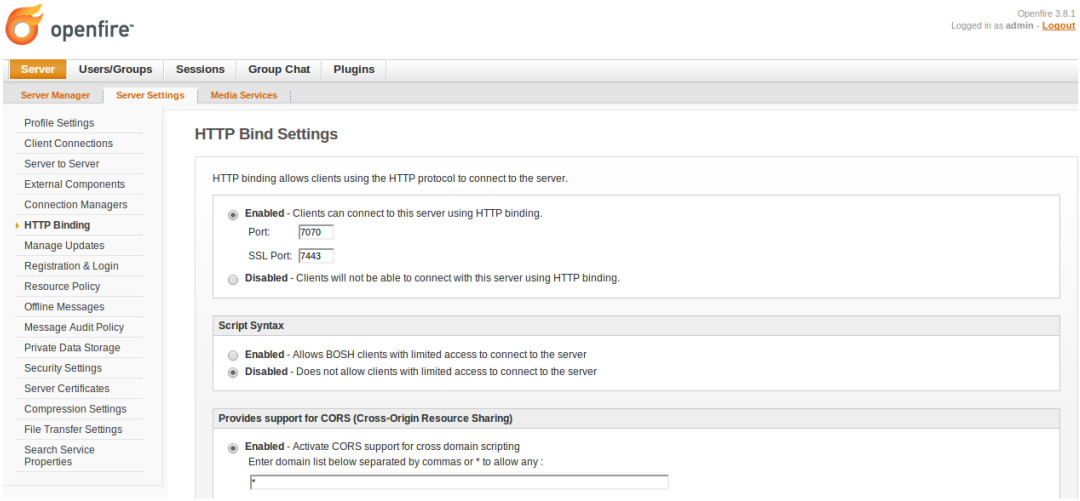


Ilustración 22 - Habilitar BOSH en Openfire

De esta manera Http-Binding queda activado en Openfire en el puerto 7070.

Por último, y se necesita activar la opción *Proxy_Http* en Apache 2. Para ello, iremos al fichero de configuración del servidor y añadiremos las siguientes normas al final del documento:

```
<VirtualHost *:80>
DocumentRoot /var/www/jabber
ServerName jabber.local
#requests sent to /http-bind will be redirected to jabber.local:7070
ProxyPass /http-bind http://jabber.local:7070/http-bind/
ProxyPassReverse /http-bind http://jabber.local:7070/http-bind/
</VirtualHost>
```

La mejor manera de comprobar que la instalación de Openfire está correctamente configurada es añadir un nuevo usuario a la red y conectarnos desde un cliente XMPP, como por ejemplo Spark podemos descargarlo gratuitamente desde http://www.igniterealtime.org/projects/spark, o incluso probando los clientes que vienen como ejemplo con la librería Strophe.

Podremos acceder a Spark con las credenciales de un usuario y la dirección del servidor, tal y como muestra la siguiente captura:

Ilustración 23 - Autenticación en Spark

Finalmente, explicaremos como añadir un nuevo usuario desde la consola de administración del servidor XMPP.

Accederemos a la consola de administración de Openfire tal y como hemos explicado anteriormente. Iremos a la administración de *Usuarios/Grupos* y desde ahí crearemos un nuevo usuario pinchando en *Crear nuevo Usuario*

Ilustración 24 - Creación de nuevo usuario en Openfire (1)

Rellenaremos todos los datos solicitados, y le clickaremos en *crear usuario*.

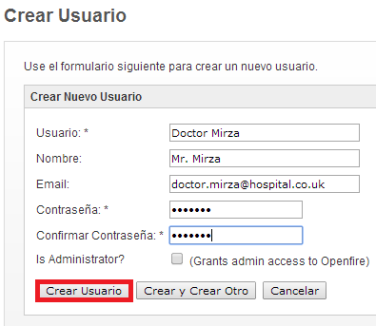


Ilustración 25 - Creación de nuevo usuario en Openfire (2)

Tras realizar estos pasos, ya habremos creado al usuario Doctor Mirza, que quedará registrado en la red y podrá acceder desde cualquier recurso simplemente conociendo el conjunto de tokens user, password y la dirección del servidor XMPP.

3.3.3 Componentes de la Red Social

Tal y como hemos avanzado en la introducción y se puede ver en el título del documento, la aplicación que vamos a crear debe de ser finalmente un gadget OpenSocial integrado en una red social.

Esto no supondrá ningún problema, ya que los gadgets OpenSocial no son otra cosa que documentos XML incluidos dentro de una web social en un especio determinado. Para servir nuestra red social utilizaremos Apache Rave.

OpenSocial

Las aplicaciones OpenSocial mantienen la misma estructura que los Google Gadgets aunque también incluyen extensiones que permiten el acceso a datos sociales (datos del usuario, listas de amigos, etc.)

Estas aplicaciones quedan alojadas en unos documentos XML con los lenguajes HTML y *javascript* integrados y pueden ser ejecutadas desde cualquier sitio web que soporte OpenSocial. En esencia son aplicaciones sencillas de crear, aunque muy potentes y con gran capacidad de desarrollo dentro de Internet.

Para comenzar a desarrollar aplicaciones OpenSocial, existen multitud de videos, multitud de manuales y multitud de documentación, aunque lo más sencillo es comenzar comprendiendo los ejemplos que incluye Apache Rave dentro de sus widgets predefinidos apoyándose en el API de OpenSocial.

Apache Rave

[11] Apache Rave es un proyecto de Apache Software Foundation que sirve como agregador de gadgets sociales o widgets web. Dicho de otra manera, será un contenedor donde agregaremos el gadget OpenSocial que hemos creado.

El objetivo de Apache Rave es convertirse en una plataforma donde usar widgets OpenSocial y W3C lo más potente posible sin ser una herramienta pesada y difícil de manipular. Se tratará de proveer de unas tecnologías y servicios que, siendo siempre fijas, permitan al usuario personalizar y contextualizar el contendor según la aplicación objetivo. Plantea también facilidades para integración con otras plataformas y soluciones.

Apache Rave provee de 11 características fundamentales:

1) Alta conformidad con OpenSocial y características de soporte opcionales
2) Soporte de widgets W3C a través de Apache Wookie
3) Persistencia OpenSocial e implementación de una SPI (Interfaz de Servicio de Proveedor)

4) Administración del portal incluyendo seguridad, gestión de gadget y plantillas de páginas
5) Gestión privada de usuario y de grupos
6) Repositorio de gadgets y gestión de los mismos y extensiones de meta-datos
7) Motor dinámico y altamente customizable para el front-end del contenedor
8) Soporte OAuth completo
9) Soporte para restricciones de seguridad en los gadgets y en el interfaz del contenedor
10) Conjunto de gadgets ejemplos de uso general
11) Soporte de mensajería inter-gadgets con ejemplos incluidos

Apache Rave, también incluye ciertas extensiones:

1) Persistencia
2) Modelo de seguridad con módulos de ejemplo para autenticación y autorización
3) Soporte para extensiones de OpenSocial futuras
4) Soporte para otros servicios de contenedores y extensiones

Con todos estos servicios que nos ofrece Apache Rave, construiremos el contenedor ideal para nuestra red social.

Instalación Apache Rave

La instalación de Apache Rave es bastante sencilla, ya que desde https://rave.apache.org/downloads.html tienes acceso a los archivos binarios del servidor, por lo que, simplemente con descargarlos y descomprimir el archivo, podremos desplegar el servidor ejecutando el archivo *startup.bat* situado en la carpeta *bin*.

Durante el despliegue del servidor (en realidad se trata de una aplicación servida en un Tomcat), la consola mostrará mensajes en forma de log con los resultados del despliegue. Habrá que tener cuidado con que no se produzca ningún error durante el mismo, aunque pueden ocurrir fallos que se solucionen con un reinicio del servidor.

Apache Rave se despliega en el puerto 8080, por lo que, antes de lanzarlo, debemos de asegurarnos de tener este puerto liberado para que se despliegue correctamente.

Para acceder a la aplicación accederemos a la URL http://localhost:8080/portal

Configuración de SQL

Por defecto, Apache Rave utiliza una base de datos H2. Para acceder a la misma basta con acceder a la aplicación que corre en el puerto 11111 a través de la URL http://localhost:11111 usando el nombre de usuario *sa* y la contraseña *local*. La base de datos H2 se guarda en los archivos temporales de Windows, en la carpeta c:/tmp/rave_db

Sin embargo, en algunas ocasiones, y a priori para nuestra aplicación, es útil configurar una base de datos basada en SQL con MySQL o PostgreSQL. De esta manera tendremos control total sobre la misma y podremos incluso compartir esta base de datos con otra aplicación. A continuación trataremos de explicar cómo configurar una base de datos externa para nuestro portal Apache Rave:

Lo primero de todo es lanzar el servidor con **extending Rave**: Para ello debemos de tener instalado Apache Maven 3.0.3 o superior, que será capaz de generar un Apache Rave custom.

Una vez hayamos lanzado Rave con sus extensiones, debemos de asegurarnos que el driver JDBC está contenido en el classpath. De esta misma manera, para que no haya conflicto entre las dos bases de datos, debemos de borrar el driver JDBC de la antigua base de datos H2. A continuación, debemos de cambiar la configuración del portal y del Shinding para que use la base de datos nueva. Las propiedades del portal las podemos encontrar en *rave-portal-resources/src/main/resources/portal.properties* y las propiedades de Shinding las podemos encontrar en *rave-providers/rave-opensocial-provider/rave-opensocial-server/rave-shindig/src/main/resources/rave.shindig.properties*.

Los parámetros de la conexión de la base de datos se deben configurar en el archivo *pom.xml* del proyecto, como aparece a continuación:

```
<!-- Rave default jdbc datasource configuration -->

<!-- The location of Rave's H2 file DB. No trailing / -->
<rave.database.location>/tmp/rave_db</rave.database.location>
<rave.dataSource.driver>org.h2.Driver</rave.dataSource.driver>
<rave.dataSource.username>sa</rave.dataSource.username>
<rave.dataSource.password>local</rave.dataSource.password>

<rave.dataSource.url>jdbc:h2:${rave.database.location};AUTO_SERVER=TRUE</rave.dataSource.url>
```

<!-- Rave Portal default jdbc datasource configuration: using rave default configuration shared with Shindig -->

<portal.dataSource.driver>${rave.dataSource.driver}</portal.dataSource.driver>

<portal.dataSource.url>${rave.dataSource.url}</portal.dataSource.url>

<portal.dataSource.username>${rave.dataSource.username}</portal.dataSource.username>

<portal.dataSource.password>${rave.dataSource.password}</portal.dataSource.password>

<!-- Rave Shindig default jdbc datasource configuration: using rave default configuration shared with Portal -->

<shindig.dataSource.driver>${rave.dataSource.driver}</shindig.dataSource.driver>

<shindig.dataSource.url>${rave.dataSource.url}</shindig.dataSource.url>

<shindig.dataSource.username>${rave.dataSource.username}</shindig.dataSource.username>

<shindig.dataSource.password>${rave.dataSource.password}</shindig.dataSource.password>

Podemos encontrar ejemplos de configuración en la web del proyecto Apache Rave en función del tipo de base de datos que usemos.

Autenticación en Apache Rave

Los servicios de autenticación de Apache Rave están manejados a través de Spring Security, que es un *framework* Java que proviene de autenticación, autorización y otros servicios de seguridad para aplicaciones empresariales.

Los mecanismos soportados por Apache Rave para la autenticación son tres: **autenticación básica** con usuarios mantenidos en una base de datos, **LDAP** y **openID**.

Autenticación básica

Se trata del clásico protocolo de autenticación en el que un cliente introduce un *username* y un *password*, que combinados, producen un mensaje codificado en *base64* que se agrega a la URL cuando el servidor manda una petición de autenticación.

Se trata de un sistema que no garantiza la confidencialidad, ya que aunque el mensaje va codificado en *base64*, no va encriptado ni comprimido en un mensaje hash, por lo que cualquier entidad C que se capture el mensaje, podría conseguir las credenciales del usuario A.

LDAP

Estas siglas pertenecen a Lightweight Directory Access Protocol y hacen referencia a un protocolo de nivel de aplicación que permite el acceso a un directorio para buscar una determinada información. También se usan estas siglas para hacer referencia a la base de datos a la que se realizan este tipo de consultas.

Este directorio tiene una estructura jerarquizada y preestablecida que queda definida en la edición del protocolo de 1993 del modelo X.500. Habitualmente estos directorios mantienen la información de *user* y *password*, pero también puede incluir datos del usuario o de sus contactos.

OpenID

OpenID[12] es un estándar abierto que permite a los usuarios ser autenticados por un RP (Relying Party), eliminando así la necesidad de incluir en la propia página web un sistema propio de autenticación y permitiendo consolidar las identidades digitales de los usuarios.

Los usuarios pueden crear sus cuentas desde su proveedores de identidades digitales preferido, pudiendo ser Google, VeriSign, Yahoo, etc. Mientras, el estándar provee de un marco para que se mantenga una comunicación entre estas entidades y los RP que permiten loguear en su sitio web.

Desde un punto de vista más técnico, en un proceso de autenticación OpenID, intervienen 3 agentes fundamentales: el usuario final, el RP (quién quiere verificar la identidad de dicho usuario) y el OpenID Provider (OP) o proveedor de identidad.

Este proceso consiste en el intercambio de un identificador único, en adelante OpenID, que será una URL o un XRI (extensible resource identifier) elegido por el usuario final para nombre se identidad. Este intercambio es activado por el agente del usuario, un navegador web.

El proceso de autenticación comienza con el agente usuario interactuando con el RP. Supongamos que previamente el usuario ya disponía de un identificador OpenID (docMirza.openid.hospital.org) registrado en un OP (gmail.com). El RP transforma el

OpenID a forma de URL canónica (http://docMirza.gmail.com) y delega en el OP la responsabilidad de autenticar al usuario. El OP realiza las operaciones necesarias para comprobar la identidad del usuario y comunica el resultado a la RP, que actuará en consecuencia, dando acceso o no al usuario al recurso solicitado. Podemos ver este proceso reflejado en el siguiente gráfico:

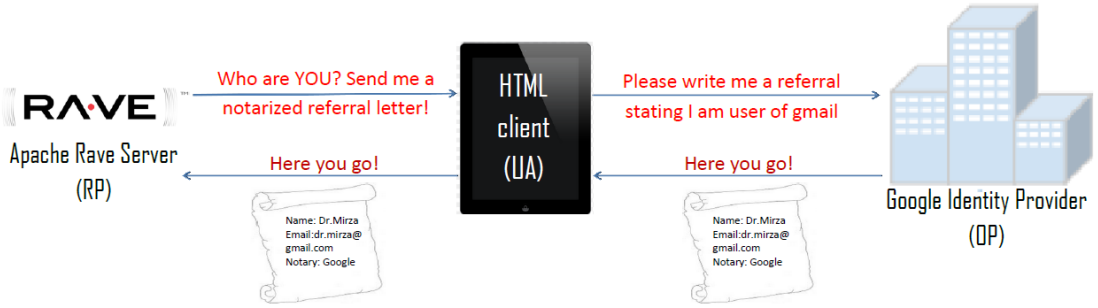


Ilustración 26 - Autenticación OpenID para Apache Rave

En el caso que nos ocupa, utilizar OpenID supondría utilizar un servidor más ligero, ya que la función de autenticación, al menos la parte computacionalmente más pesada, queda delegada a una OP. Para realizar esta función, podemos pensar en dos opciones. La primera y más simple sería utilizar un OpenID identifier servido por Google: esto aportaría sencillez al proyecto, pero en contra, el dominio del identificador quedaría fuera de nuestro alcance, y por lo tanto, también la gestión de los identificadores y de los mecanismo de autenticación; la segunda sería montar nuestro propio servidor OpenID, definiendo el dominio y gestionando los usuarios directamente. A priori no parece complicado montar este servidor, ya que existen algunos opensource tal como **WSO2 Identity Server**, compatible con OpenID entre otras tecnologías.

Incluir aplicaciones en Apache Rave

Para incluir aplicaciones en Apache Rave, la mejor manera es loguearse en el portal a través de la URL http://localhost:8080/portal (supongamos que lo estamos ejecutando en local) con el usuario canonical {user:canonical, password:canonical} ya que es el administrador por defecto del sistema.

Una vez estamos en el portal, iremos al *Widget Store*.

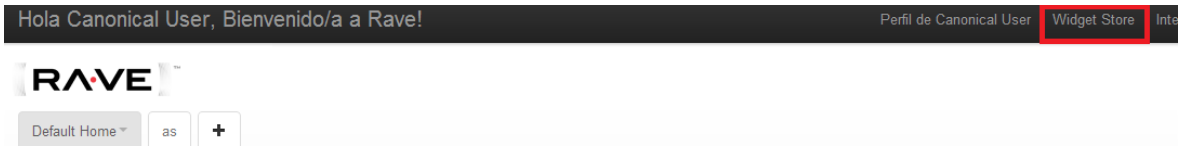


Ilustración 27 - Insertar nuevo widget en Rave (1)

A continuación, pincharemos sobre *Agregar nuevo Widget*.

Ilustración 28 - Insertar nuevo widget en Rave (2)

Rellenamos la localización de nuestro widget (siempre ha de estar dentro de la carpeta demogadgets) y pulsamos en *Obtener meta-data del widget.*

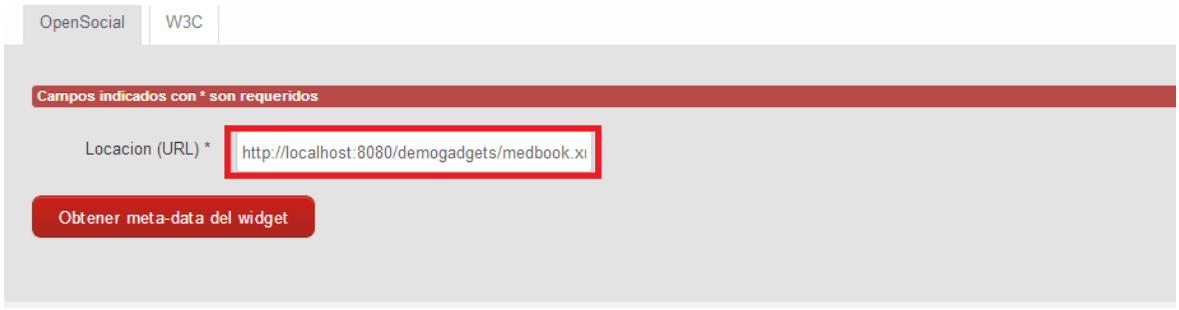


Ilustración 29 - Insertar nuevo widget en Rave (3)

Rellenamos los metadatos de nuestro widget

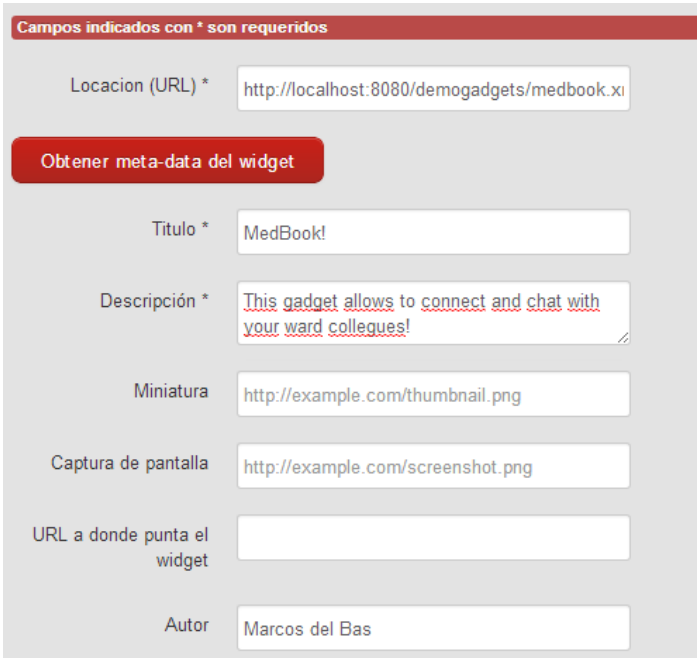


Ilustración 30 - Insertar nuevo widget en Rave (4)

Y finalmente pinchamos en *Agregar widget*.

Ilustración 31 - Insertar nuevo widget en Rave (5)

Una vez realizados estos pasos, el widget ya se encuentra subido en la *Widget Store*, pero aún falta tomar una acción más para que este widget pueda se añadido a una página de Rave.

Iremos al *Interfaz de Administración*

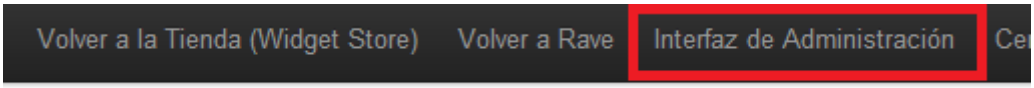


Ilustración 32 - Insertar nuevo widget en Rave (6)

Y allí iremos a *Widgets* para poder administrar la nueva aplicación instalada.

Ilustración 33 - Insertar nuevo widget en Rave (7)

Buscaremos nuestra aplicación (están ordenadas por orden alfabético) y pincharemos en ella.

Dentro de las opciones del widget, cambiaremos el *Estatus* de PREVIEW a PUBLISHED

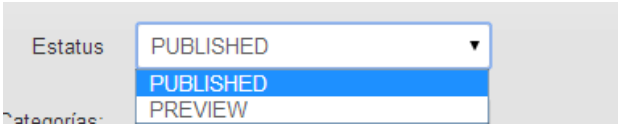


Ilustración 34 - Insertar nuevo widget en Rave (8)

Ahora ya si podremos añadir nuestro Widget desde la *Widget Store*.

3.3.4 Conclusiones acerca del diseño

Hemos visto cuáles son las funciones del servidor Openfire, así como las distintas opciones que tenemos en él para configurarlo y servir la aplicación XMPP. A su vez, hemos entendido cuales son las virtudes de XMPP, qué ventajas nos ofrece y cuáles son los complementos que mejoran sus servicios.

Por otro lado, hemos experimentado con Apache Rave lo suficiente para ser capaces de integrar una aplicación en un widget y poder agregar al portal de la red social este widget. También hemos visto algunos de los complementos y funcionalidades extras que nos ofrece este servidor.

De todas las posibilidades que nos ofrecen estas dos herramientas, hay dos en concreto, de las que ya hemos hablado, que nos pueden ayudar bastante a la integración completa del chat XMPP en la red social como un gadget OpenSocial: XML- javascript widgets y la posibilidad de utilizar bases de datos externas en ambas aplicaciones y relacionarlas entre sí.

Ambas posibilidades facilitarían, por un lado, la integración de la aplicación XMPP en la plataforma de Rave como widget y, por otro, la gestión de los usuarios de ambos servidores, de manera única.

Como hemos podido observar durante todo el documento, las tecnologías que hemos escogido nos permitirán sobradamente implementar un sistema que nos permita comunicar a los usuarios de una red social a través de una aplicación funcional, segura y escalable. Todas ellas son tecnologías muy utilizadas en la actualidad, tanto en pequeños grupos desarrolladores como en grandes empresas, lo cual permite la continuidad del proyecto más allá del objetivo inicial. Y por encima de todo, se tratan de componentes y librerías opensource, pudiendo ser utilizadas por cualquier persona sin necesidad de poseer una licencia.

Además, XMPP está pensado para que los clientes sean navegadores, lo cual nos permite llegar a todo tipo de usuarios, sea cual sea el recurso tecnológico del que dispongan. Dicho de otra manera, cualquier usuario podrá conectarse desde un laptop, un PC, un Mac, un teléfono móvil Android, iOS o Windows Phone, ya que todos estos gadgets incorporan en sus sistemas operativos al menos, un navegador para ejecutar la aplicación. Así mismo, la filosofía de XMPP permite mantener abierta una conexión por largo tiempo, evitando costosos sistemas de polling por parte del cliente, lo que se traduce en ahorro de batería y menor consumo del servicio de la aplicación en el cliente.

Este último hecho, y la idea que impulsa el desarrollo de esta aplicación hace que las posibilidades de expansión de la misma sean reales, ya que podría utilizarse en muchos campos profesionales en los que hoy en día se carece de este tipo de comunicación, como puede ser la medicina, y en concreto, el mundo que rodea la vida de un hospital. Siendo todo el staff técnico (enfermeros, médicos, cirujanos, terapeutas, psicólogos, etc.) un posible objetivo para la implantación de este tipo de sistemas.

Finalmente, simplemente tener en cuenta el uso de OpenSocial como valor añadido en esta aplicación, ya que nos permite acudir a datos sociales (listas de amigos, nombres de usuarios, etc) fácilmente usando las APIs proporcionadas por esta tecnología.

3.4 Implementación

En este sub-capítulo realizaremos una descripción de los cambios realizados en la implementación de la aplicación a lo largo de todo el desarrollo del proceso.

3.4.1 Fase de inicio

Para la implementación de la aplicación utilizaremos *javascript* apoyándonos en la librería *Strophe* para la implementación de características XMPP y en la librería *jQuery* para tomar acciones sobre los elementos de la página web que realicemos.

En esta primera versión, buscaremos implementar una aplicación muy sencilla en la que un usuario sea capaz de enviar mensajes a otros y se muestre un roster estático sin información de presencia alguna.

En la siguiente imagen podremos ver el aspecto que tiene esta primera versión de la aplicación en la que, con capacidades reducidas respecto a una versión final, tratamos de resolver el principal problema planteado en este proyecto, la comunicación entre los usuarios.

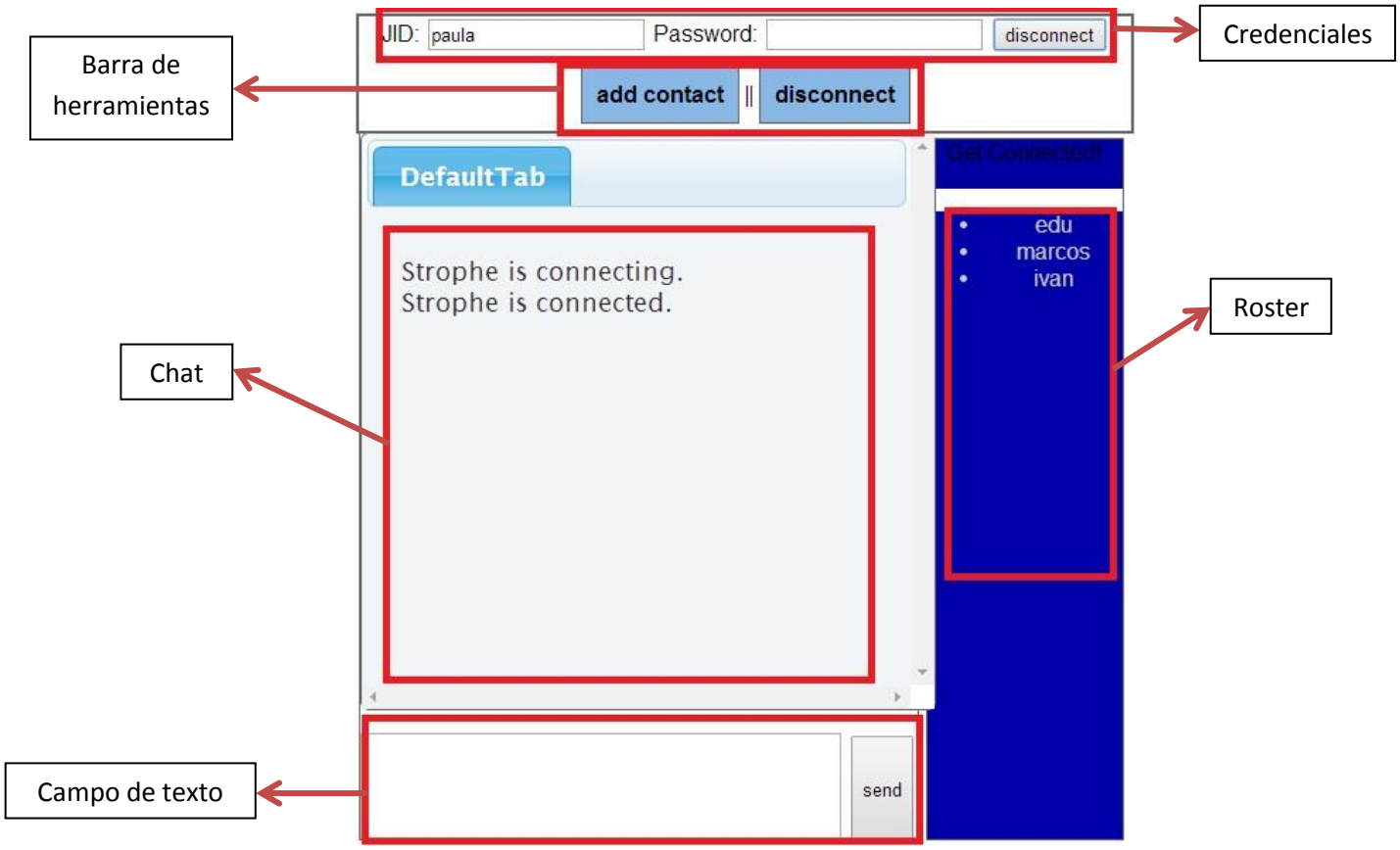


Ilustración 35 - Interfaz gráfica de la aplicación en la fase de inicio

Como se puede ver en la imagen, hay 5 elementos principales dentro de nuestra aplicación: Credenciales, barra de herramientas, roster, chat y campo de texto.

Credenciales: Este campo está compuesto por un formulario con 2 entradas, en las que incluiremos nuestro JID (sin incluir el dominio de la red XMPP) y la contraseña para ese JID; y un botón para conexión/desconexión del usuario de la red según el estado de la aplicación. Al rellenar las dos entradas con un JID y contraseña correctas y presionar el botón *conectar*, la aplicación, a través de la librería *Strophe* mandará una directiva XMPP para conectarse al servidor Openfire. En todo momento, el estado de la conexión es mostrado en el campo Chat en la pestaña *DefaultTab*.

Barra de herramientas: Esta barra incluye dos botones, aunque en esta versión solamente el botón de desconexión está activo. Al pulsar este botón, la aplicación, de nuevo apoyada en *Strophe*, envía una directiva al servidor Openfire para desconectarse de la red XMPP. En el transcurso de la desconexión, el estado de la aplicación queda reflejado en la pestaña *DefaultTab*.

Chat: Este campo incluye un elemento de tipo *tab* que se gestiona a través de la librería *jQueryUI*. Se pueden incluir tantas pestañas como el número de usuarios con el que se quiera chatear. El campo incluye una barra de desplazamiento para que el texto siga apareciendo cuando se llegue al final del campo. Cuando un nuevo mensaje llega de un cliente, se agrega un nuevo elemento * * a este chat incluyendo el contenido del mensaje.

Roster: La funcionalidad de este elemento de la página web es básicamente mostrar los amigos del cliente de la aplicación. Como hemos dicho anteriormente se trata de un roster estático, por lo que no actualiza la información de presencia (por eso aparecen todos los nombres en blanco) ni las nuevas amistades. Este campo se basa en una lista desordenada que agrega tantos elementos como amigos tenga el cliente de la aplicación, los cuales se obtienen enviando una petición *iq:roster* al servidor Openfire.

Campo de texto: Este elemento sirve como entrada de texto para el envío de mensajes a otros clientes. Agrega un botón para el envío del mensaje. En caso de que el campo de texto esté vacío, simplemente no se envía ningún mensaje. En caso contrario, se genera una *stanza* XMPP a través de una directiva de *Strophe* de tipo *message* que se envía al servidor para que vaya a parar al cliente objetivo. Aunque existan varias pestañas, en esta versión, por simplicidad, sólo se permite mantener comunicación con el primer amigo que sale en la lista.

Todos estos elementos quedan integrados en la aplicación y son servidos desde un cliente HTTP que se conecta a un servidor Openfire ayudándose de BOSH para que la conexión se mantenga abierta sin necesidad de realizar polling desde el cliente.

Por otro lado, necesitaremos desplegar el servidor Openfire para que se puedan conectar los usuarios al mismo. En un primer momento, este servidor se despliega en local para realizar pruebas sobre él. Se configura el servidor para que BOSH quede activado y se gestionen las desconexiones de los usuarios tras un tiempo inactivos. En un primer momento instalaremos la versión 3.8.2 del servidor.

3.4.2 Fase de desarrollo

De base tomamos la versión 1.0 de nuestra aplicación y buscamos soluciones sobre la misma, intentando no cambiar la estructura de la aplicación. A su vez, buscamos la

mejor manera de encajar las nuevas funcionalidades a añadir a la aplicación sin cambiar la tecnología sobre la que nos estamos apoyando.

De nuevo dejaremos a un lado el servidor Apache Rave por simplicidad a la hora de hacer pruebas. Mantenemos la relación entre el servidor Openfire y los clientes que se conectan a la aplicación.

Dividimos este apartado a su vez en dos sub-capítulos distintos: por un lado, se realiza una nueva implementación de la interfaz gráfica de nuestra aplicación para hacerla más atractiva e intuitiva. Por otro lado, se realizan los cambios pertinentes para agregar las nuevas funcionalidades requeridas y diseñadas en los anteriores apartados.

En cuanto a la interfaz gráfica incluimos nuevos elementos en nuestra aplicación para permitir intercambiar información de presencia entre los clientes, y la inclusión de nuevos amigos a nuestra red de contactos. Por otro lado, se reestructuran los elementos de la página para hacer más atractiva e intuitiva la aplicación.

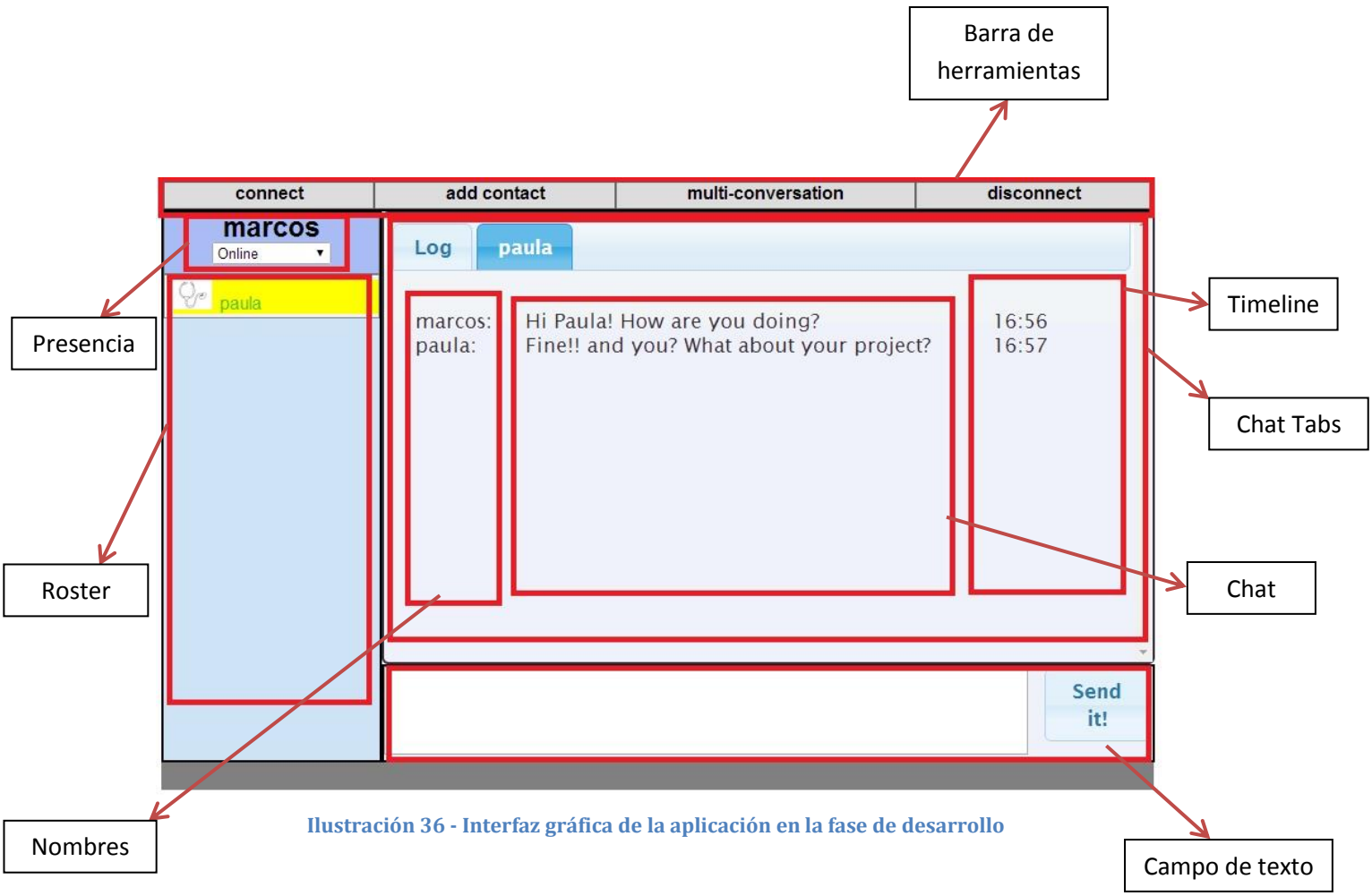


Ilustración 36 - Interfaz gráfica de la aplicación en la fase de desarrollo

De los 5 elementos que teníamos en la versión anterior, hemos conservado 4 de ellas, hemos ampliado una y hemos añadido otra.

Barra de herramientas: Hemos añadido dos botones más a nuestra barra de herramientas. El botón *connect* nos permitirá conectarnos a la red tras habernos desconectado, eso sí, será con el mismo usuario con el que anteriormente estábamos conectados. El botón *multi-conversation* se crea para generar una sala para conversar con varios usuarios de manera concurrente, aunque en esta versión, no tendrá ninguna funcionalidad y se dejará como posible mejora futura de nuestra aplicación. Por último el botón para agregar nuevos usuarios funciona, por lo que al presionarlo aparece el siguiente diálogo web:

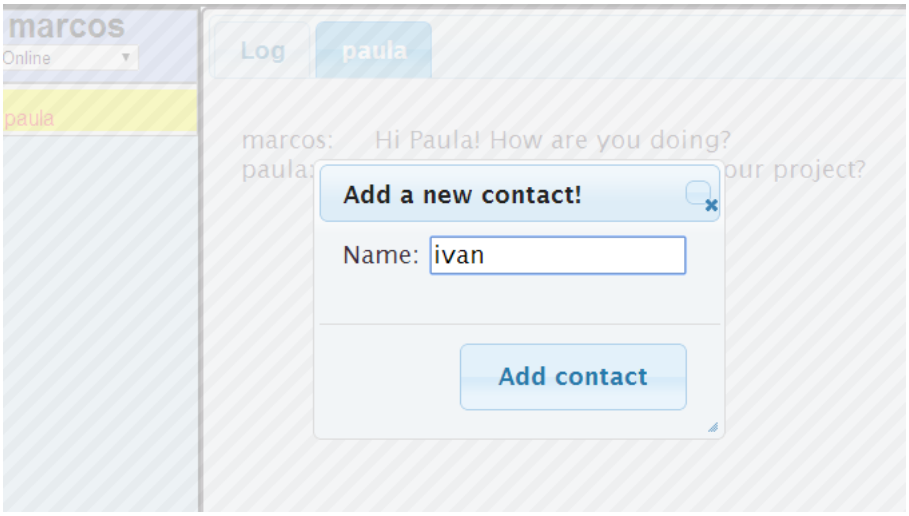


Ilustración 37 - Añadiendo un nuevo contacto

En relación a esta herramienta para añadir contactos, cabe destacar el proceso para agregar amigos a la red XMPP. Consiste en una doble suscripción al servicio de presencia por parte de los dos usuarios, es decir, para que el usuario A esté suscrito al servicio de presencia de B, A debe solicitarlo a B y B debe de aprobar esta solicitud por parte de A. De esta manera la suscripción sería de tipo *both*. Esta situación queda implementada en nuestra aplicación por medio de un diálogo web que se activa al recibir una suscripción.

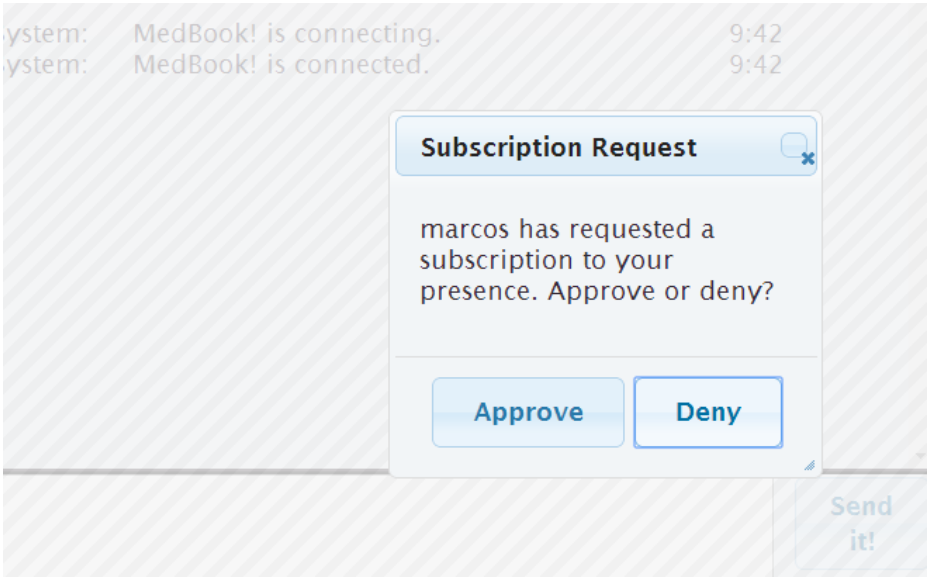


Ilustración 38 - Aprobación de suscripción

Llegados a este punto, podremos aceptar o rechazar la petición de suscripción. Si la rechazamos, la aplicación no sufrirá cambio alguno y el roster quedará intacto. En caso contrario, el roster deberá actualizarse, quedando de la siguiente manera.

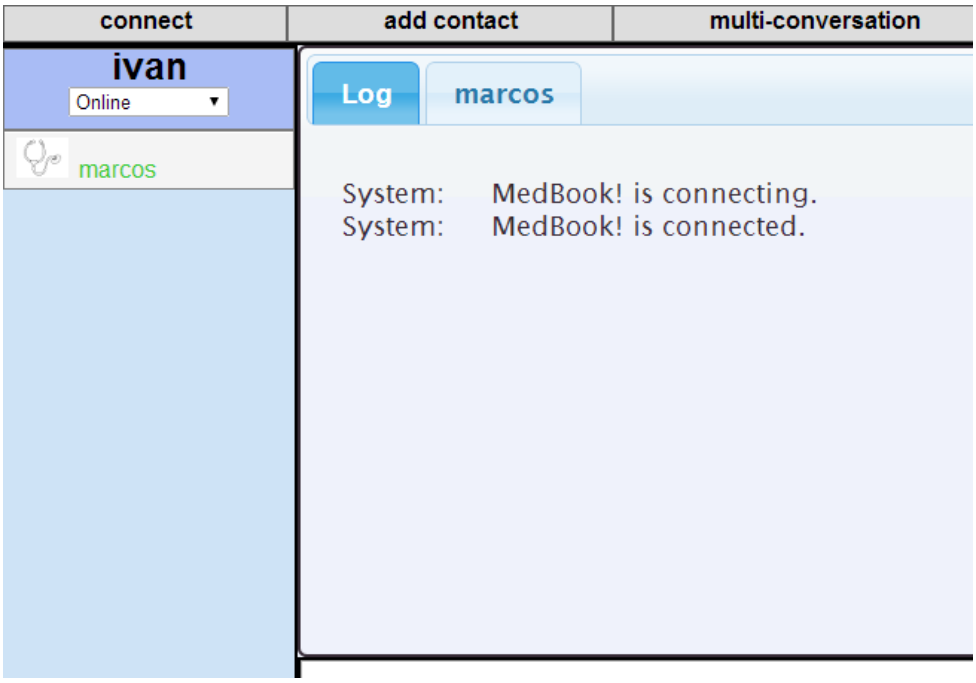


Ilustración 39 - Nuevo amigo añadido al roster

Presencia: Este elemento se ha añadido en esta versión para poder enviar información de presencia a nuestros amigos dentro de la red XMPP. Se trata de una lista

desplegable que dispone de 3 estados “Online”, que quedará reflejado en verde en nuestro roster; “Away”, que quedará reflejado en color amarillo; y “Do not disturb” que quedará reflejado en rojo. Por otro lado, cuando un usuario esté desconectado, sus amigos verán reflejado este estado en gris en su roster.

Chat Tabs: Este elemento se ha mejorado bastante para hacerlo más atractivo e incluir más información en el mismo. Se ha mejorado el sistema de pestañas, de tal manera, que al presionar en uno de nuestros contactos del roster se activa la pestaña de dicho contacto. Se han incluido dos nuevos sub-elementos como son **Timeline**, que en esencia es una línea temporal para reflejar la hora a la que se entregan los mensajes, y los **nombres** que especifican que usuario envía cada mensaje.

Roster: De nuevo este campo vuelve a ser esencial en nuestra aplicación, y en este caso además de mostrar la información de los amigos que tiene el usuario de la aplicación en forma de lista, añade la funcionalidad de cambiar el color del usuario en función de su información de presencia, completa el elemento con color amarillo cuando un nuevo mensaje del usuario llega e incluye la posibilidad de poner una imagen al lado de cada usuario en la lista de amigos (aunque a modo de ejemplo todos tengan la misma imagen).

Campo de texto: Este elemento permanece invariante desde la primera versión. Mantiene el campo de texto y el botón para enviar los mensajes. Se enviarán los mensajes al usuario cuya pestaña esté activa.

Además de todos estos elementos, para conectarse, al iniciar la aplicación, aparece otro diálogo donde se puede introducir las credenciales.

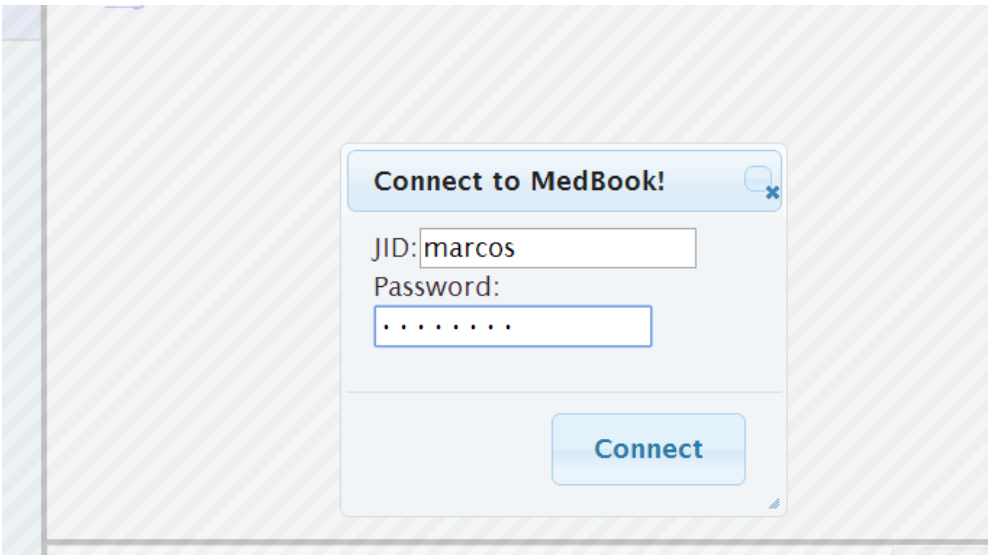


Ilustración 40 - Acceso a la aplicación mediante credenciales

Como se puede ver, con todos los cambios realizados en la aplicación, la interfaz ha mejorado bastante su aspecto. Ahora se muestra la información de una manera más completa y más clara.

Por otro lado, se han incluido todas las funcionalidades requeridas en el planteamiento inicial de esta fase.

3.4.3 Fase de integración

Como ya hemos explicado en el apartado de análisis y diseño, los cambios a realizar en esta fase son: integración de la aplicación en un gadget XML, autenticación única y gestión única de los usuarios manejada por el cliente. Trataremos cada punto por separado y explicaremos cómo hemos implementado estos cambios.

Integración de la aplicación en un gadget XML: Analizamos algunos gadgets de ejemplo que se incluyen dentro de Apache Rave para conocer la estructura que deben de tener este tipo de aplicaciones. Vemos que se tratan de archivos XML en los que se incluyen unas directivas para cargar los metadatos de la aplicación y utilizan la sección CDATA de estos documentos para incluir el código HTML. Por tanto, en principio sólo se trata de incluir dentro de esta sección nuestro código HTML, cargar los metadatos propios de nuestra aplicación y cargar este gadget en el contenedor de Apache Rave (podemos ver como se carga un gadget dentro de Apache Rave en el capítulo de *estado del arte* de esta misma mermoria).

Realizamos lo cambios pertinentes en nuestro nuevo documento XML incluyendo también el código CSS dentro del mismo (ya que si no, no se cargaba la hoja de estilos correctamente), nos loggeamos en Apache Rave e incluimos el nuevo gadget. La página web tiene la siguiente apariencia:

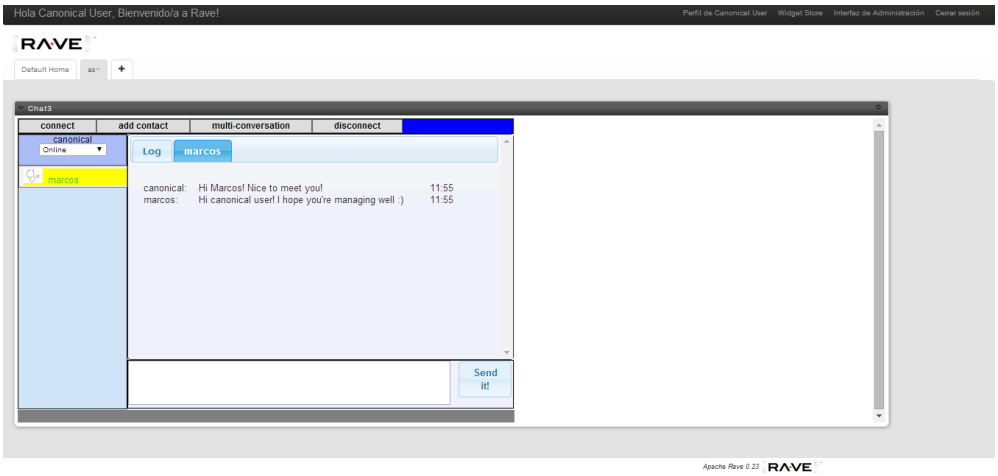


Ilustración 41 - Apariencia del gadget instalado en Rave

Como se puede ver, la aplicación es capaz de conectarse desde la red social al servidor XMPP con un usuario nuevo, como es el usuario canonical, y este usuario puede agregar a otros, al igual que podíamos hacer anteriormente (en el ejemplo capturado agregamos al usuario "Marcos"), y podemos mantener una conversación con el mismo. Nuestra aplicación se adapta perfectamente a Apache Rave y este es capaz de servirla sin ningún problema.

Autenticación única: Hemos comentado anteriormente, que para implementar este hito el cliente debe de gestionar las credenciales del usuario conectado a la red social y lanzar una petición al servidor XMPP para conectarse al mismo. Explicaremos a continuación este proceso en 4 pasos más simplificados:

1) El cliente, nada más ejecutarse, manda una petición al servidor Apache Rave utilizando el API de OpenSocial pidiéndole el nombre del usuario que está ejecutando el gadget.

```
var req = opensocial.newDataRequest();
req.add(req.newFetchPersonRequest(opensocial.Id
Spec.PersonId.VIEWER), 'viewer');
req.send(onLoadCredentials);
```

El cliente manda la petición y captura los resultados obtenidos, que será el nombre del usuario que ha accedido a Apache Rave.

2) Una vez el cliente tiene el nombre del usuario que se ha conectado a la red social, generará una petición HTTP que lanzará al servidor Openfire (previamente debemos de haber instalado el plugin *User Service*) para

añadir a este cliente como nuevo usuario de la red XMPP. En esta petición incluirá el nombre de usuario, una contraseña generada a partir de este nombre y una clave secreta que le permita acceder a los cambios de la consola de administración del servidor Openfire. El cliente lanza la petición al puerto 9090 del servidor.

3) El servidor recibe la petición y comprueba que la clave secreta es válida. Si esta es válida procede a comprobar los datos incluidos en la petición. Pueden ocurrir dos cosas: si la petición está completa y el cliente es nuevo, se genera este nuevo cliente en la red XMPP; si la petición está incompleta o está completa pero el cliente no es nuevo, no se hará ningún cambio en la lista de clientes y se mandará una respuesta específica para cada casuística.

4) El cliente HTML recibe la respuesta a la petición del servidor. En caso de respuesta negativa, el cliente no hace nada. En caso de respuesta positiva, procede a conectar la aplicación al servidor Openfire con las credenciales del cliente creado anteriormente.

Gestión única de los usuarios: Una vez la aplicación tiene capacidad para generar nuevos clientes en la red XMPP desde el cliente HTML, el siguiente paso es conseguir que la lista de amigos de ambos servidores sea la misma. Para ello, volveremos a hacer uso de las peticiones dirigidas desde el cliente hacia ambos servidores. Será un proceso similar al anterior, aunque las peticiones que se generen será distintas. Lo explicaremos también en 4 pasos más simplificados:

1) A la vez que el cliente HTML manda la petición OpenSocial al servidor Apache Rave para conocer los datos del cliente, manda otra petición al mismo servidor para conocer la lista de amigos del mismo:

```
var viewerFriends = opensocial.newIdSpec({"userId": "VIEWER",
"groupId": "FRIENDS"});
 var opt_params = {};
opt_params[opensocial.DataRequest.PeopleRequestFields.MAX] =
100;
req.add(req.newFetchPeopleRequest(viewerFriends, opt_params),
'viewerFriends');
req.send(onLoadCredentials);
```

De nuevo el cliente mandará la petición al servidor de la red social y capturará en una lista los amigos del usuario.

2) Este paso será prácticamente el mismo que el que tomamos cuando hablábamos de autenticarnos de manera única, a diferencia de que en este caso, no será una petición, si no varias, las que mandaremos al puerto 9090 del servidor Openfire, ya que la lista de amigos puede llegar a ser de 100 usuarios (hemos impuesto esta limitación como prueba, aunque llegado el caso se podría aumentar). Iremos cogiendo cada elemento de la lista de amigos y mandaremos una petición por amigo al servidor.

3) El servidor Openfire comprobará que cada petición cumple los requisitos necesarios y que la clave secreta es correcta. De nuevo, en función de si la petición es correcta o no, y si los amigos ya estaban registrados en la base de datos de la red XMPP o no, el servidor responderá de una manera u otra al igual que en la anterior ocasión.

4) Finalmente, el cliente recibe las respuestas a tantas peticiones como amigos tenga en la red social y manda una *stanza iq:roster* para actualizar el roster al servidor XMPP. De esta manera, en el roster de la aplicación quedarán registrados los cambios y aparecerán los amigos del usuario en la red social.

Con todos estos cambios implementados, tenemos nuestra aplicación perfectamente integrada como gadget en Apache Rave, pudiendo disfrutar al completo de todas las funcionalidades que habían sido implementadas en fases anteriores y ofreciendo al usuario una experiencia integral, evitando duplicidades y dobles autenticaciones innecesarias.

4. Pruebas realizadas

4.1 Introducción

Las pruebas se realizaron como una fase más dentro de cada iteración del proyecto, aunque en este documento sólo se mostrarán las realizadas tras la fase de integración, ya que carecían de relevancia durante las otras fases.

Con el fin de probar las diferentes capacidades de la aplicación implementada, dividimos el trabajo en 5 áreas.

- Pruebas de interfaces y contenidos
- Pruebas de funcionalidades y operación
- Pruebas de carga
- Pruebas de seguridad
- Pruebas de respaldo y recuperación

4.2 Contenidos de las pruebas

4.2.1. Pruebas de interfaces y contenidos

Las pruebas de esta etapa consisten en hacer revisiones específicas para el despliegue de contenidos de las páginas de la aplicación, ver si se cumplen los *Términos de Referencia* en estos temas y si se cumplen los estándares mínimos.

Las pruebas que realizamos en esta etapa serán:

Verificación de contenidos: Es una prueba básica en la que se busca conocer si la aplicación contiene todos los términos especificados en el plan de desarrollo de la aplicación. Esta prueba se puede realizar de forma manual o automática aunque nosotros la haremos de forma manual a través de la navegación por las páginas de la aplicación.

Verificación de estándares: En esta prueba tratamos de comprobar si la aplicación cumple los estándares del lenguaje de programación utilizado (HTML, CSS y javascript en nuestro caso), así como las normas establecidas para ellas en la

organización del código fuente. Para el caso de XML, HTML y CSS utilizamos validadores externos (World Wide Web Consortium).

Validación de interfaces: Mediante esta prueba revisamos los aspectos gráficos del sitio para comprobar si se despliegan correctamente. Los elementos que incluimos en la verificación serán:

- **Consistencia de la diagramación:** Con el fin de ofrecer al usuario una experiencia similar en cualquier sitio de la aplicación, la página debe tener elementos consistentes.
- **Ancho de diagramación:** Comprobar si los contenidos tienen el ancho adecuado para el tamaño predeterminado de la aplicación.
- **Diagramación vs. Browsers:** Comprobaremos si los contenidos se muestran correctamente en varios navegadores: *Mozilla, Google Chrome* e *Internet Explorer* (aunque se espera que no funcione correctamente)
- **Diagramación vs. Sistemas Operativos:** Comprobaremos si los contenidos se muestra correctamente en varios S.O: *Windows 7*, *Linux*, *Android* e iOS.
- **Imágenes escaladas:** Se debe comprobar que las imágenes mostradas no estén siendo escaladas por programación.

4.2.2 Pruebas funcionales y de operación

En estas pruebas se realizan chequeos completos respecto a las funcionalidades y aplicaciones de nuestro sitio. Las pruebas se deben hacer sobre diferentes elementos.

Validación de formularios: Si el sitio web contiene formularios para el envío o ingreso de datos se deben utilizar sistemas de validación de ingresos de datos para comprobar que estos sean bien ingresados. Las validaciones que hacemos sobre los formularios serán:

- **Campos obligatorios:** los campos obligatorios han de estar marcados de alguna manera (asterisco usualmente) que permitan a los usuarios conocer que los datos han de ingresarse de forma obligatoria
- **Validaciones locales:** para evitar la sobrecarga del servidor se ha de incorporar la mayor parte de estas en el cliente.

- **Sintaxis de ingreso:** los datos de ingresos deben de ser válidos.
- **Ingreso de datos:** si los datos ingresados van a parar a una base de datos, debe comprobarse en el campo de destino que los datos introducidos son válidos.
- **Multiplataforma:** se debe comprobar que los formularios funcionan en diferentes plataformas.

4.2.3 Pruebas de carga

Para las pruebas de carga utilizamos pruebas de carga realizadas sobre un servidor Openfire con clustering activado.

4.2.4 Pruebas de seguridad

Las pruebas de seguridad van enfocadas a comprobar si los accesos a la aplicación son seguros para la privacidad del usuario.

Mecanismos de control de accesos: incorporación de mecanismos modernos de generación de claves y autenticación como puedan ser la firma electrónica o la autenticación usuario y clave.

Roles mínimos a asegurar: deben de existir una serie de roles para asegurar el seguro y correcto funcionamiento de la aplicación, en nuestro caso se validarán:

- **Administrador de la aplicación:** encargado del funcionamiento del software
- **Administrador de seguridad:** encargado del cumplimiento de las directivas de seguridad
- **Administrador de contenidos:** encargado de las informaciones contenidas en la web

4.2.5 Pruebas de rapidez de acceso

Para realizar estas pruebas establecemos una lista de chequeo con respuesta simple (SI/NO). En caso de no pasar el chequeo, se deben de anotar las acciones correctivas a tomar en el futuro.

4.2.6 Pruebas de usabilidad

El test de usabilidad de un sitio, nos permite:

- Verificar la existencia de posibles problemas de usabilidad en el sitio.
- Encontrar posibles soluciones para los problemas encontrados.
- Establecer una medida concreta inicial contra la cual comparar a los competidores, futuros desarrollos de este mismo sitio o modificaciones al actual.

El test de usuarios es el tipo de evaluación más importante y la mayor herramienta de desarrollo posible para un sitio web.

Para realizar este test estableceremos una lista de chequeo con el fin de obtener los aspectos a corregir. La respuesta deberá de ser SI/NO y en caso negativo, deberán de anotarse las medidas correctivas.

4.3 Resultados de las pruebas

A continuación mostramos los resultados obtenidos al aplicar las pruebas que anteriormente hemos explicado en nuestra aplicación en su versión final.

4.3.1 Resultados de pruebas de interfaces y contenidos

Prueba	Resultado
Verificación de contenido	Se ha comprobado y verificado todos los contenidos especificados en el desarrollo manualmente.
Verificación estándar XML	Verificado en http://validator.w3.org/
Verificación estándar HTML	Verificado en http://validator.w3.org/
Verificación estándar CSS	Verificado como parte del código XML en http://validator.w3.org/

Consistencia de la diagramación	La diagramación es consistente desde la propia definición de la aplicación
Ancho de la diagramación	Los elementos no se descolocan al reducir la resolución
Diagramación vs. Browsers	Como era esperado, la diagramación se mantiene para *Mozilla* y *Google Chrome*, pero no para *Internet Explorer*
Diagramación vs. Sistemas Operativos	Se realiza la prueba en *Linux* y *Windows 7* con resultados positivos. Imposibilidad de realizar la prueba en Android e iOS
Imágenes escaladas	Las imágenes no están escaladas

Tabla 11 - Resultados de pruebas de interfaces y funcionales

4.3.2 Resultados de pruebas funcionales y de operación

Prueba	Resultado
Campos obligatorios	Se especifican los campos que son obligatorios
Validaciones locales	La única validación se realiza en el servidor y no en el cliente
Sintaxis de ingreso	Los datos ingresados han de ser válidos para proceder a la autenticación de los clientes
Ingreso de datos	Los datos no se validan al ingresarse ya que son datos de autenticación
Multiplataforma	Los formularios se comportan bien para los navegadores *Mozilla Firefox* y *Google Chrome*

Tabla 12 - Resultados de pruebas funcionales y de operación

4.3.3 Resultados de pruebas de carga

Desde el grupo de desarrollo del servidor Openfire se vienen realizando pruebas de stress al servidor para probar su capacidad, buscando los límites de Openfire.

En la actualidad estos test de stress permiten conectar 500.000 usuarios a único servidor Openfire cambiando frecuentemente el estado de presencia, enviando mensajes y añadiendo usuarios a sus listas de amigos. La CPU del servidor funcionando entre un 30-50% de su capacidad y consumiendo 16Gb de memoria RAM.

Las características del servidor sobre el que se han realizado estos test de stress son: SPARC Enterprise T5220, 32GB RAM, CPU - UltraSPARC-T2 con 8 procesadores y 8 hilos en cada procesador, velocidad CPU - 1165MHz, 146GB 10k HDD SAS y SCSI.

Existe la posibilidad de habilitar clustering para los servidores Openfire, lo que permite disminuir el consumo de CPU.

4.3.4 Resultados de pruebas de seguridad

Prueba	Resultado
Autenticación con usuario y clave	Existen procedimientos para recuperar la clave en el caso de que a un usuario se le pierda o no la recuerde.
Roles mínimos	Se garantiza la existencia de un administrador de la aplicación y de un administrador de seguridad (Apache Rave). La presencia de un administrador de contenidos no se garantiza por completo.

Tabla 13 - Resultados de pruebas de seguridad

4.3.5 Resultados de pruebas de rapidez de acceso

Pregunta	Respuesta
¿El usuario puede encontrar en no más de cuatro clics la información buscada?	SÍ
¿Aparece el menú de navegación en un lugar destacado? ¿Se ve fácilmente?	SÍ
¿El sitio cuenta con un mapa y/o buscador que dé un acceso alternativo a los contenidos?	NO

¿Es fácil llegar a las secciones más importantes del sitio desde cualquier página?	SÍ
¿El sitio mantiene una navegación consistente y coherente en todas sus páginas?	SÍ
¿El diseño usa jerarquías visuales para determinar lo importante con una sola mirada?	Sí
¿Los formularios ofrecen opciones que permitan al usuario evitar, cancelar o rehacer una acción?	NO
¿El tamaño de la letra de los textos es adecuado y ajustable o modificable por el usuario usando las herramientas del navegador?	SÍ
¿Los vínculos, imágenes e iconos son claramente visibles y distinguibles?	SÍ
¿Los vínculos (links) visitados y no visitados son claramente diferenciables?	NO
¿Los iconos son representativos de la función o acción que realizan y son aclarados mediante un 'title'?	NO
¿Todas las páginas cuentan con información de contactos virtuales y/o físicos al pie de página?	SÍ
¿El nombre de la URL está vinculado con el nombre del sistema y se ofrece en la barra superior del navegador?	NO
¿Ofrece el sitio contenidos sobre la visión, misión y objetivos?	NO
En el caso de que existan palabras técnicas en los contenidos del sitio, ¿existe una sección de glosario que las explique?	NO

¿Ofrece páginas de ayuda que explican cómo usar el sitio web?	NO
En caso de errores de consistencia dentro del sitio, ¿se ofrece mensaje personalizado mediante una página explicativa?	NO

Tabla 14 - Resultados de pruebas de rapidez de acceso

4.3.6 Resultados de pruebas de usabilidad

Pregunta	Respuesta
Identidad corporativa	
¿La portada del sitio refleja la identidad y pertenencia de la entidad?	SÍ
¿Existen elementos de la imagen corporativa de su sitio? ¿Se repiten en todas las páginas?	NO
¿El logotipo ha sido incluido en un lugar importante en la Portada y en las páginas interiores del sitio?	NO
¿Todas las páginas cuentan con un título que indique el nombre de la institución e información de contactos virtuales y físicos al pie de la página?	SÍ
Utilidad del sitio web	
¿El sitio ofrece información sobre las actividades y servicios más recientes e importantes que está llevando a cabo?	NO
¿Los usuarios pueden encontrar fácilmente en la portada la información acerca de las actividades y servicios más importantes	NO

de la entidad?	
Navegación	
¿El diseño del sitio es eficiente, rápido e intuitivo?	SÍ
¿Aparece el menú de navegación en un lugar destacado? ¿Se ve fácilmente?	SÍ
¿Verificó la consistencia de todos los enlaces?	SÍ
¿El sitio cuenta con un mapa o buscador que facilite el acceso directo a los contenidos?	NO
¿El sitio mantiene una navegación consistente y coherente en todas las pantallas?	SÍ
Visibilidad del estado del sistema	
¿Se informa al usuario claramente el área del sitio que está visitando?	SÍ
¿El sitio web diferencia entre enlaces visitados y enlaces por visitar?	NO
Consistencia y cumplimiento de estándares	
¿El HTML del sitio ha sido validado satisfactoriamente según w3c.org?	SÍ
¿Las hojas de estilo (CSS) han sido aprobadas según w3c.org?	SÍ
Atención de errores	
¿Usa javascript para validar formularios durante su llenado y antes de enviarlos?	NO

¿Después de que ocurra un error, es fácil volver a la página donde se encontraba antes de que se produjese o entra recomendaciones de los pasos a seguir?	NO
Estética y diseño	
¿Usa jerarquías visuales para determinar lo importante con una solo mirada?	SÍ
¿Las imágenes tienen tamaños adecuados que no dificultan el acceso a las páginas?	SÍ
¿Las imágenes tienen etiqueta para facilitar la navegación	NO
Ayuda ante errores	
En caso de errores de consistencia dentro del sitio, ¿se ofrece un mensaje personalizado mediante una página explicativa?	NO
¿Ofrece área de preguntas frecuentes con datos de ayuda a usuarios?	NO

Tabla 15 - Resultados de pruebas de usabilidad

4.4 Valoración de las pruebas realizadas

A la vista de los resultados obtenidos, se puede concluir que las pruebas más importantes se han pasado con éxito, pero que aún pueden existir mejoras en materia de seguridad, ayuda ante errores y contenido aclaratorio del sitio.

Cabe destacar que la aplicación en su conjunto cumple con los estándares establecidos por la W3C y que la estructura de la diagramación permite una experiencia sencilla al usuario.

5. Evaluación

En esta fase del proyecto, trataremos de evaluar si los requisitos establecidos en el desarrollo del producto se han cumplido o no, así como las causas o razones de los mismos, y a su vez estableceremos si los objetivos marcados en el inicio del proyecto se han llevado a cabo y por tanto la aplicación es útil dentro del marco establecido.

Para realizar la evaluación del producto, enfocaremos cada requisito establecido de manera independiente, y estudiaremos su cumplimiento. La misma acción se realizará con los objetivos marcados al inicio del proyecto.

5.1 Evaluación de requisitos

REQ_1: El sistema debe de estar formado por elementos de código abierto

Para analizar el cumplimiento del requisito, recordaremos cada elemento que compone nuestra aplicación:

Apache Rave: Se trata de un motor web que sirve como contenedor para agregar widgets a una red social. Apache Rave da soporte para widgets OpenSocial a través de la integración del API OpenSocial. Se trata de un proyecto bajo el copyright de *The Apache Software Foundation* en el que colaborar distintos grupos de la empresa. El código se puede descargar desde la misma página web del proyecto https://rave.apache.org/, para ser compilado y ejecutado desde Linux o Windows.

Openfire: Este servidor utiliza el protocolo XMPP para dar varios servicios, entre ellos, el principal: la mensajería instantánea. Integra también un portal de administración desde el que gestionar a los usuarios y grupos de la red, así como distintas opciones sobre plugins o servicios extras. Se trata de un software libre desarrollado por *Jive Software* bajo la licencia Apache 2.0. Tanto la última versión del servidor, como los plugins del mismo, se pueden descargar de manera gratuita desde la página web del pryecto http://www.igniterealtime.org/projects/openfire/

Strophe: Strophe.js es una librería XMPP escrita para desarrollar código para dicho protocolo en javascript. Incorpora funciones propias que facilitan el uso del protocolo

XMPP. Esta implementación está preparada para clientes tipo web y necesita de BOSH para funcionar con Openfire, lo cual nos permite conectar las llamadas HTTP al protocolo XMPP a través de conexiones duraderas en el tiempo, evitando así sistemas basados en polling. El acceso a la documentación de la librería, así como el acceso al código de la misma se puede encuentrar en http://strophe.im/strophejs/

User Service plugin: *User Service* es un plugin para Openfire que nos permite gestionar los usuarios a través de peticiones HTTP mandadas desde el cliente a la consola de administración de Openfire. Se trata de un plugin para este servidor al que se puede acceder de manera gratuita desde http://www.igniterealtime.org/projects/openfire/plugins.jsp

Cliente web: Para ejecutar la aplicación cliente, así como para acceder a la red social necesitamos disponer de un navegador web para conectarnos a los servidores que sirven nuestra aplicación. La misma está preparada para funcionar totalmente tanto en *Google Chrome* como en *Mozilla Firefox* y parcialmente en *Internet Explorer*. Los dos primeros navegadores se pueden descargar gratuitamente desde las páginas web de sus desarrolladores http://www.google.com/intl/es/chrome/browser/ y http://www.mozilla.org/es-ES/firefox/new/

REQ_2: El sistema ha de permitir mensajes de gestión

Para realizar el complimiento de este requisito, hemos **permito** dos tipos de mensajes de gestión:

Por un lado, la propia naturaleza del protocolo XMPP, permite que se intercambien mensajes de gestión de estado entre los distintos clientes a través de las *stanzas* tipo *iq* y *presence*. Este es un servicio que ofrece XMPP y que hemos aprovechado en nuestra aplicación para gestionar los roster y para indicar la disponibilidad de los clientes.

Por otro lado, para resolver un problema de conflictos entre los dos servidores de los que hace uso la aplicación, se ha habilitado un mecanismo para que sea el cliente quién gestione los usuarios y sus listas de amigos entre ambos servidores. El cliente obtendrá los datos del servidor Apache Rave, y a través de peticiones HTPP generará cambios en la consola de administración de Openfire. De esta manera, evitaremos tener listas de amigos que generen conflicto entre ambos servidores, ya que siempre serán las mismas.

REQ_3: El sistema debe de tener una funcionalidad de red social

De nuevo XMPP ha sido nuestra gran ventaja para cumplir este requisito. Queda cubierto con los propios servicios que ofrece XMPP, que recordemos son:

- **Encriptación del canal:** Ofrece encriptación en la comunicación cliente-servidor y en la comunicación servidor-servidor. Nos permite crear aplicaciones seguras.
- **Autenticación:** Todas las entidades de la red XMPP han sido anteriormente autenticadas por un servidor, que actúa, en este caso, como un guardián de la red.
- **Presencia:** Este servicio nos permite conocer si una entidad está o no disponible en cualquier momento de su conexión, pero además, nos permite incluir información adicional tal como la razón de su indisponibilidad, si se encuentra en una reunión, etc. Para obtener la información de presencia de un contacto, se ha de estar suscrito a dicho contacto, protegiendo la privacidad del usuario.
- **Lista de contactos:** El uso más común de este servicio, es el de otorgar al usuario la capacidad de iniciar una conversación rápidamente con cualquier otro usuario al que esté suscrito.
- **Mensajes one-to-one:** El uso tradicional en la red de este servicio es la Mensajería Instantánea. Sin embargo, en esencia se trata de un intercambio de información XML, por lo que puede ser también usado entre bots, o entre servidores, etc.
- **Mensajes multiusuario:** Este servicio nos permite utilizar salas, similares a las del mítico IRC, en las que se intercambian mensajes que pueden ser texto plano, configuración de la sala, mensajes de control de la sesión, etc.
- **Notificación:** Permite generar notificaciones y enviarlas a todas aquellas entidades suscritas.

De estos tres servicios, los básicos, y los cuales hemos implementado para cumplir los requisitos de nuestra aplicación son: Autenticación, presencia, lista de contactos y mensajes one-to-one.

REQ_4: La red social debe de ser capaz de servir otros futuros gadgets

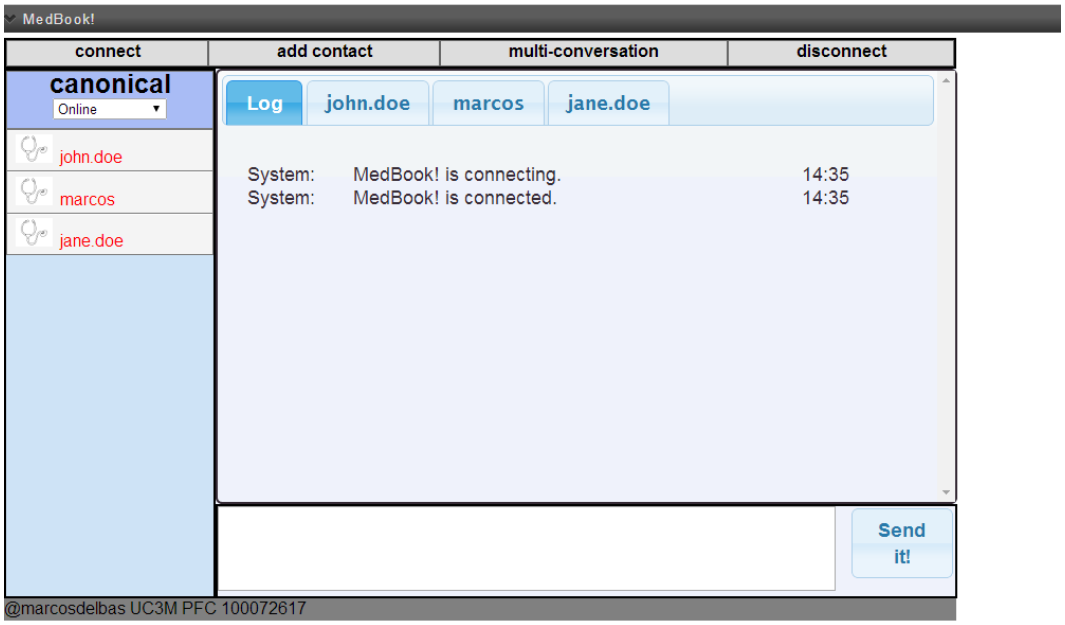
Como contenedor para servir el gadget que hemos implementado hemos escogido Apache Rave. Como ya hemos explicado varias veces a lo largo del documento, Apache Rave nos permite agregar gadgets desde un documento XML (el proceso para agregarlo se puede ver completo en el punto 2 de esta memoria).

Apache Rave permite mantener una biblioteca de gadgets para que los usuarios puedan escoger entre ellos cuáles son los que quieren tener instalados en sus páginas de la red social. Estos gadgets pueden ser actualizados automáticamente sin necesidad de recargarlos en el servidor.

Por tanto, Apache Rave nos permitirá incluir otros gadget OpenSocial en el futuro que complementen la aplicación que hemos implementado con nuestro proyecto ya sean tablones de noticias, cuadros de estadísticas dinámicas, etc.

REQ_5: La Interfaz gráfica debe ser simple y funcional

Este requisito quizás sea el que más tiempo ha llevado cumplir. Sin embargo, el resultado final cumple perfectamente con los requisitos. Como podremos ver en la siguiente imagen, la interfaz gráfica de la aplicación web es sencilla e incluye todas las funciones que se han ido requiriendo a lo largo del proyecto.



REQ_6: El sistema debe de tener una gestión única de usuarios

Este fue el último requisito que se incorporó a la lista y lo hizo en la última iteración de la fase de desarrollo del proyecto ya que la duplicidad de la gestión de los usuarios fue un problema que surgió tras estudiar y analizar cuales iban a ser los elementos que iban a conformar la aplicación en su conjunto.

Esta duplicidad de usuarios (Apache Rave mantiene su propia lista de amigos del usuario y el servidor Openfire hace lo mismo para la red XMPP) puede provocar varios

problemas. El primero y más importante es que la lista de amigos de la red social no se corresponda en todo momento con la lista de amigos en la red XMPP servida por Openfire. Por otro lado, pudiera resultar, que aun correspondiéndose, ambas listas no se actualicen cuando la otra lo haga.

Ambos problemas quedan resueltos gracias al plugin *User Service* de Openfire. Este plugin queda instalado en el servidor (se pueden gestionar sus opciones desde la consola de administración del mismo, desde donde también se puede instalar) y permite gestionar a los usuarios y las listas de los usuarios sin necesidad de acceder a la consola de administración del servidor. En concreto, se puede acceder a ella lanzando una petición HTTP hacia la consola de administración (situada en el puerto 9090).

A través de estas peticiones y del API OpenSocial, la aplicación cliente es capaz de solicitar la información del usuario y sus listas de usuarios al servidor Apache Rave y de transmitir estas listas al servidor Openfire para que realice los cambios pertinentes en estas listas si es necesario. De esta manera conseguimos evitar que los usuarios sean gestionados por dos entidades distintas.

5.2 Evaluación de Objetivos

Tras el análisis de los hitos cumplidos a lo largo del proyecto se puede concluir que todos los objetivos que se han marcado se han cumplido con mayor o menor éxito, pero adaptándose la consecución de los mismos a los recursos reales de los que se disponía.

5.2.1 Objetivo principal

Al inicio del proyecto se fijó como objetivo principal el de "habilitar una herramienta software que facilite la comunicación entre los agentes del mundos hospitalario y que abra la puerta para mejorar la asistencia sanitaria en pacientes externos a través del control de los mismos a través de una red social".

Analizando estas palabras, se puede concluir que se ha concluido con éxito el objetivo principal. Hemos conseguido integrar una herramienta en la red social Apache Rave que permite conectarse con los amigos que tengamos en la red social para comunicarse entre sí. Además, el soporte de la red social permite que en el futuro se

implementen otros gadgets para el control de los pacientes o para mejorar las comunicaciones entre el personal. Así mismo, la herramienta integrada está basada en un protocolo de comunicación en auge y que permite integrar aún más servicios como la videoconferencia o la multi-conversación.

5.2.2 Objetivos secundarios

En cuanto a estos segundos objetivos, se fijaron dos durante el inicio del proyecto: el primero hablaba sobre "que la herramienta pueda ser utilizada desde cualquier dispositivo portátil" y el segundo "que la herramienta sea de fácil uso y manejo".

En cuanto al primero de los objetivos, se ha tratado de buscar una tecnología que permitiera ejecutar la herramienta desde cualquier dispositivo, ya sea portátil o fijo. La plataforma sobre la que se ejecuta la aplicación es una red social, a la que se puede acceder desde cualquier equipo que disponga de un navegador. Por tanto, el objetivo ha sido cumplido. Sin embargo, este objetivo pudiera ser interpretado de otra manera que permitiera un acceso más simple y más cercano a las opciones actuales como pueda ser una aplicación móvil para una plataforma determinada (ya sea Android o iOS) que permitiera acceder a la red social sin necesidad de utilizar el navegador. Esta opción permitiría una mayor flexibilidad para los usuarios de la aplicación y facilitaría el acceso a la misma, y por ello se estudiará más adelante como una ampliación del proyecto.

En relación al segundo y último de los objetivos, durante las últimas fases del proyecto se ha realizado un gran esfuerzo para que la interfaz de la aplicación fuera lo más sencilla posible de cara a la usabilidad y a su vez se ha conseguido que, para que el acceso para el usuario sea lo menos costoso posible, el gadget se integre perfectamente en la red social gracias al API de OpenSocial y al plugin para Openfire *User Service*, que juntos nos permiten acceder a los datos sociales de los usuarios y a su vez lanzar peticiones al servidor Openfire para que añada nuevos usuarios y/o modifique sus listas.

6. Aportaciones personales y valoración

En este capítulo se tratará de reflejar todas aquellas aportaciones que este proyecto me ha dado, así como el proceso de aprendizaje que me ha llevado a adquirir determinados conocimientos.

Por otro lado se tratará de valorar el trabajo realizado a lo largo de este proyecto y durante cada una de sus fases.

6.1 Aportaciones

Como una de las últimas partes de este proyecto se pretende describir las aportaciones personales que se han obtenido a lo largo del desarrollo del mismo.

La realización de este proyecto fin de carrera me ha aportado valiosos conocimientos en el mundo de la mensajería instantánea y las redes sociales. Estos conocimientos forman una parte esencial dentro de mi formación como Ingeniero de Telecomunicación y espero poder aplicarlos y aumentarlos en el futuro.

El primero y más importante de ellos es el conocimiento de un protocolo de comunicación en auge y con gran demanda en la actualidad como es XMPP. A lo largo de los últimos años, no ha pasado un día sin que utilizara *Whatsapp* o *GTalk*, y sin embargo, aunque intuía su funcionamiento, desconocía por completo la tecnología que hacía funcionar a estas dos aplicaciones tan demandadas. XMPP, además de ser el protocolo más utilizado para mensajería instantánea, es un protocolo sencillo pero muy potente. Ofrece una gran cantidad de servicios (podríamos decir que los ofrece todos dentro del mundo de la mensajería instantánea) que permiten construir aplicaciones con grandes funcionalidades y posibilidades. Así mismo, me ha permitido conocer cómo se gestiona un servidor XMPP como es Openfire con capacidad para cientos de miles de usuarios concurrentes.

Como segunda aportación a mi formación, la realización de este proyecto me ha permitido conocer qué es un API Social y más concretamente qué es OpenSocial: quién está detrás de este proyecto y las pretensiones que hay detrás de él. Pienso que OpenSocial puede ser una API muy utilizada en el futuro debido a las grandes empresas que lo impulsan. Sin embargo, me ha decepcionado encontrarme con tantas dificultades para encontrar información y foros de desarrolladores activos. La idea que hay detrás de OpenSocial es muy buena, pero sin embargo, no se le da todo el impulso que debería de

dársele a este conjunto de APIs, ya que son realmente valiosas para los desarrolladores. Así mismo, creo que aún se trata de un proyecto poco maduro, y la demostración de ello es que el contenedor de gadgets que utilizamos para cargar la aplicación y a su vez como red social, que es Apache Rave, está aún en su versión beta.

Por otro lado, la implementación de la aplicación me ha permitido refrescar los conocimientos que tenía de HTML que adquirí en mi época de instituto y ampliar mis conocimientos de programación con javascript. De esta manera, me he adentrado en el mundo de las aplicaciones web y en más concreto en el desarrollo de widgets para redes sociales. Esto también me ha permitido conocer jQuery, la cual considero una herramienta imprescindible para la programación de aplicaciones web dinámicas.

Como condición para la realización de la aplicación, era la utilización de HTML y javascript para la implementación. Por ello decidí buscar una librería javascript para XMPP, y me topé con Strophe. Aprender a utilizar esta librería ha ampliado los conocimientos que tenía sobre XMPP y además, me ha permitido conocer BOSH, una herramienta sin la que sería imposible que la aplicación funcionara.

Finalmente, como parte del proceso de aprendizaje a lo largo de este proyecto, ha habido un elemento que considero también muy importante. Este hecho ha sido aprender por primera vez como se desarrolla un proyecto desde su nacimiento a su final.

6.2 Valoración

Durante todas las fases de la realización del proyecto, la valoración del trabajo realizado ha sido positiva. Así mismo, las aptitudes demostradas han sido suficientes para la realización con éxito del trabajo, y la actitud tomada durante el tiempo de realización del proyecto ha sido la correcta para la consecución del mismo.

Echando la vista atrás, se pueden vislumbrar varias fases distintas en las que la valoración ha de hacerse por separado.

En una primera fase, en la que se ha organizado la estructura del proyecto y se han estudiado las partes teóricas necesarias en fases posteriores, el trabajo ha sido más lento, aunque la dedicación era a tiempo parcial, debido a la cantidad de documentación que había que estudiar y comprender. Sin embargo, ese trabajo realizado lo considero el más importante, ya que me ha permitido avanzar con mayor rapidez y menor cantidad de

errores en las siguientes fases, y a su vez, me ha permitido avanzar con la documentación del proyecto en una fase temprana.

En la fase de implementación y desarrollo del proyecto el trabajo realizado ha ido *in crescendo* a lo largo de todo el proceso. La primera versión de la aplicación llevó más tiempo realizarla que la segunda versión, a pesar de las características reducidas de esta versión, y al igual que en la fase anterior, la paciencia para la implementación correcta de las funcionalidades ha permitido que la segunda versión fuera realizada más eficazmente.

Por último, en la fase de mantenimiento y pruebas finales el trabajo realizado ha sido muy positivo ya que se ha podido comprobar el correcto funcionamiento de la aplicación y el cumplimiento de los estándares sin necesidad de arreglar muchos errores.

En general, estoy muy satisfecho con el trabajo realizado. Por encima de todo he comprendido que para gestionar un proyecto completo se ha de tener paciencia y que las primeras fases de documentación y diseño son básicas para la ejecución exitosa del proyecto.

7. Conclusiones y trabajo futuro

En este último capítulo trataremos de describir las conclusiones que hemos sacado a lo largo de la realización del proyecto y trataremos de dibujar las líneas de trabajo futura que consideramos más adecuadas.

7.1 Conclusiones

En este proyecto se ha pretendido mostrar las posibilidades reales de una aplicación basada en OpenSocial para mensajería instantánea alojada dentro de una red social.

Como se ha visto a lo largo del proyecto, se ha tratado de buscar siempre la solución más adecuada con vistas al futuro de la aplicación. De esta manera, dejamos la puerta abierta a otros proyectos que puedan surgir a partir de este que lo complementen y lo mejoren.

La decisión de utilizar XMPP como tecnología base para la aplicación de IM ha sido realmente acertada, ya que hemos podido contar con una tecnología potente que nos ofrece una gran cantidad de servicios y que nos ha permitido utilizar una liberería como es Strophe que nos ha ayudado bastante durante la implementación. Además, esta elección permitirá que el proyecto no quede obsoleto rápidamente, sino todo lo contrario, permitirá el desarrollo de la aplicación más allá de este proyecto concreto.

Así mismo, utilizar un API social para integrar las características de una red social dentro de nuestra aplicación como es OpenSocial nos ha permitido ampliar el campo de acción del proyecto y abrir las puertas a nuevos complementos futuros.

Cabe destacar el uso en todo momento de herramientas de software libre. Nos proporciona una mayor cantidad de posibilidades debido a la cantidad de información de la que se dispone de este tipo de herramientas. Sin embargo, hay que ser cuidadoso con la documentación que se utiliza pues en ocasiones puedes encontrar con que las fuentes de ésta sean poco fiables. Por tanto, al utilizar este tipo de herramientas siempre conviene buscar en fuentes oficiales, y cuando no es posible, contrastar la información antes de utilizarla para el desarrollo del proyecto.

Finalmente hay que tener en cuenta las posibilidades que ofrecen los servicios de la aplicación dentro del campo en el que se quiere aplicar, como es la medicina. Es evidente que el apoyo de la tecnología es más que necesario para la mejora continua de los servicios prestados por Sanidad, y la utilización de aplicaciones como la que se ha desarrollado en

este proyecto puede mejorar los servicios que se prestan a los ciudadanos y a su vez mejorar la productividad de los trabajadores.

7.2 Trabajo Futuro

A lo larga de la memoria hemos ido dejando entrever cuales pueden ser las líneas a seguir trabajando en relación a este proyecto. De todas las que se han estudiado se van a proponer aquellas que se consideran más interesantes para el éxito del proyecto.

- **Ampliación de las funcionalidades de la aplicación:** Como hemos podido estudiar, XMPP nos permite ofrecer más servicios de los que actualmente se sirven en la aplicación. Cuatro de los servicios que consideramos importantes para implantar en un futuro serían:
  - **Multiconversación:** Permitiría a los usuarios crear salas de chat donde se pudieran unir otros usuarios mediante invitación. De esta manera se podrían discutir temás que involucren a varios equipos de trabajo.
  - **VoIP:** No consideramos que sea un servicio esencial para cubrir nuestra aplicación, sin embargo puede resultar muy interesante para algunas situaciones concretas en las que el usuario necesita una respuesta rápida o no pueda utilizar las manos en ese momento.
  - **Videoconferencia:** Este servicio permitiría a los usuarios de la aplicación atender a formaciones y realizar cursos online impartidos por otros usuarios de la red.
  - **Transferencia de archivos:** Puede resultar interesante que se compartan archivos de un equipo de trabajo a otro cuando el paciente necesite de ambos.
- **Creación de aplicaciones para dispositivos móviles:** Aunque la aplicación puede ser utilizada desde cualquier dispositivo que disponga de un navegador, la experiencia de usuario es mejor cuando estos servicios se dan desde una aplicación nativa. Se proponen dos sistemas operativos móviles para la implantación de estas aplicaciones: Android e iOS.
- **Creación de nuevos gadgets OpenSocial:** Con el objetivo de articular una serie de herramientas alrededor de la que se ha creado en este proyecto, se propone implementar nuevos gadgets utilizando el API de OpenSocial para

complementar los servicios ofrecidos por la aplicación. Se proponen dos gadgets en concreto:

- **Tablas de equipos y pacientes:** Este gadget mostraría una tabla con los equipos que trabajan con los pacientes que el usuario de la aplicación tiene asignados. De esta manera, se podría conocer en tiempo real las profesionales que trabajan con un paciente determinado.
- **Resultados de pruebas y seguimiento de pacientes:** Este gadget permitiría acceder a los resultados de las pruebas que se realizan a los pacientes y también al seguimiento de los datos de los pacientes hospitalizados externamente.

- **Seguimiento de contenedores OpenSocial:** Sería también importante contar con la posibilidad de realizar un seguimiento a aquellos contenedores OpenSocial que puedan ir apareciendo en el mercado buscando la posibilidad de migrar la aplicación a otro contenedor con mejores características y más actualizado.

Referencias

[1]. Peter Saint-Andre , Kevin Smith, Remko Tronçon "XMPP: The Definitive Guide", O'Reilly Media, abril 2009

[2]. Jack Moffitt “Professional XMPP Programming with Javascript and JQuery”, Wiley Publishing Inc, 2010

[3]. XMPP Standards Foundation www.xmpp.org

[4]. Strophe library www.strophe.im

[5]. BOSH over Openfire + Strophe http://expertnotfound.wordpress.com/2013/04/08/setting-up-openfire-bosh-strophe/

[6]. jQuery Documentation www.jquery.com

[7]. jQueryUI Documentation www.jqueryui.com

[8]. Openfire Server Documentation www.igniterealtime.org/projects/openfire

[9]. OpenSocial Foundation Documentation www.opensocial.org

[10]. OpenSocial Wiki https://opensocial.atlassian.net/wiki/display/OSD/Specs

[11]. Apache Rave Documentation www.rave.apache.org

[12]. OpenID Foundation Specifications http://openid.net/developers/specs/

[13]. Using Zephyr MIT https://sipb.mit.edu/doc/zephyr/

[14]. Mumble www.mumble.com

[15]. SIP protocol RFC http://www.ietf.org/rfc/rfc3261.txt

[16]. Mozilla Social API https://developer.mozilla.org/es/docs/Social_API

[17]. HybridAuth API http://hybridauth.sourceforge.net/

[18]. Google+ API https://developers.google.com/+/api/

[19]. OneAll API http://www.oneall.com/

Glosario

En orden alfabético:

ACK	ACKnowledgement
AIM	AOL Instant Messenger
AJAX	Asynchronous JavaScript And XML
AOL	American Online
API	Application Programming Interface
AT&T	American Telephone & Telegraph
BBS	Bulletin Board System
BHI	Beverly Hills Internet
BOSH	Bidirectional-streams Over Synchronous HTTP
CPIM	Common Presence and Instant Messaging
CPU	Central Processing Unit
CSS	Cascading Style Sheets
CTSS	Compatible Time Sharing System
DOM	Document Object Model
HTML	HyperText Markup Language
IBM	International Business Machines corporation
ICQ	I seek you
IESG	Internet Engineering Steering Group
IETF	Internet Engineering Task Force
IM	Instant Messaging
iOS	Anteriormente iPhone Operating System
IP	Internet Protocol
IRC	Internet Relay Chat
JDBC	Java DataBase Connectivity
JID	Jabber ID
jQuery UI	jQuery User Interface
JSON	JavaScript Object Notation
LDAP	Lightweight Directory Access Protocol
MIM	Mobile Instant Messaging
MIT	Massachusetts Institute of Technology
MSN	Microsoft Messenger Service
MySpaceIM	MySpace Instant Messenger
NAT	Network Address Translation
OP	Openid Provider
P2P	Peer-to-peer
Q-Link	Quantum Link
RAM	Random Access Memory
REST	REpresentational State Transfer
RFC	Request For Comments
RP	Relying Party

RRSS	Redes Sociales
SIMPLE	Session initiation protocol for Instant Messaging and Presence Leveraging Extensions
SIP	Session Initiation Protocol
SQL	Structured Query Language
TCP	Transmission Control Protocol
UA	User Agent
UAC	User Agent Client
UAS	User Agent Server
UDP	User Datagram Protocol
UNIX	Deformación de UNICS: Uniplexed Information and Computing System
URL	Uniform Resource Locator
VoIP	Voice over IP
W3C	World Wide Web Consortium
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol
XRI	Extensible Resource Identifier

Anexo A: Manual de Usuario

La aplicación *Medbook!* te permite conectarte con tus compañeros de trabajo para chatear y compartir información.

A.1 Instalación

Para instalar la aplicación debes acceder a tu red social Apache Rave en la dirección http://163.117.141.214:8080/portal/login y usar tus credenciales para acceder.

Ilustración 42 - Inicio de sesión en Rave para instalación

En caso de no disponer de una cuenta en Apache Rave, puedes crear una nueva desde la misma URL accediendo al enlace que hay en la parte superior derecha “Cree una cuenta nueva”.

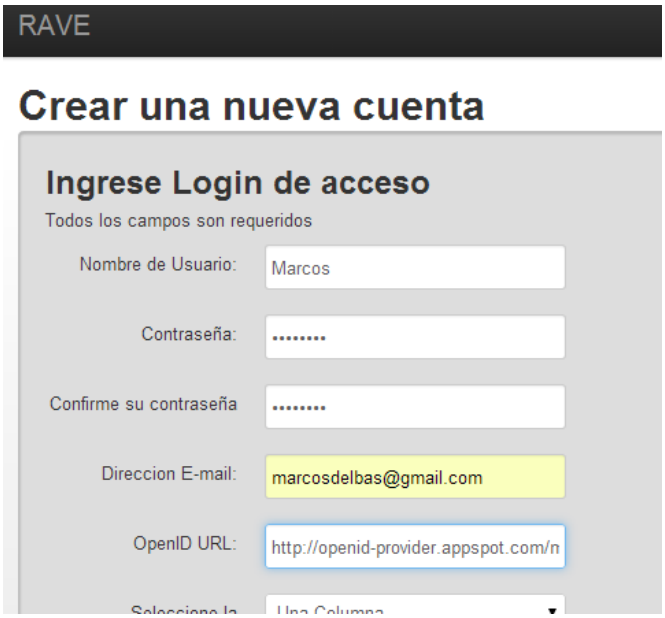


Ilustración 43 - Creación de una cuenta en Apache Rave

Una vez dispongamos de una cuenta en Apache Rave, accederemos al mismo con nuestras nuevas credenciales.

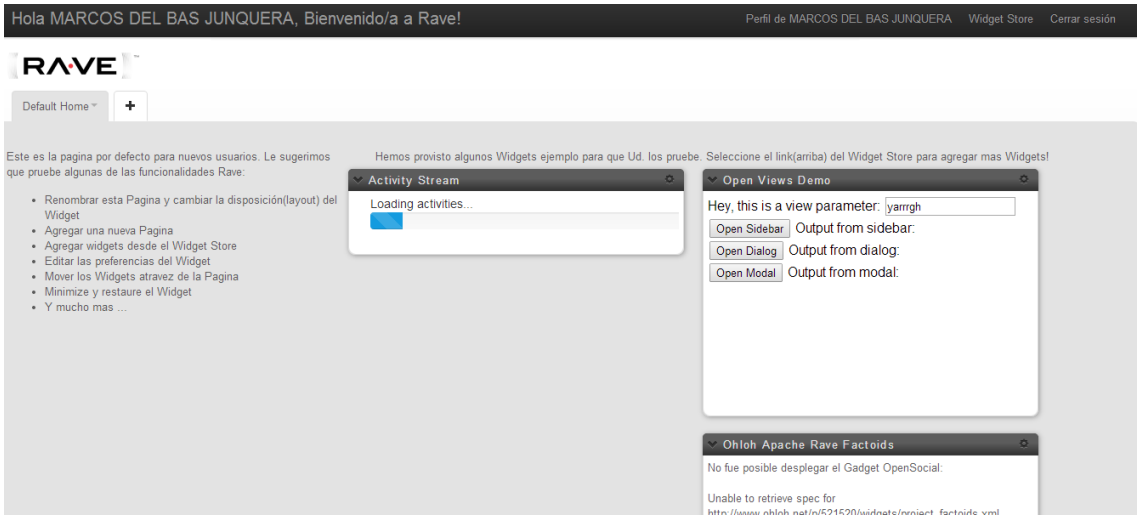


Ilustración 44 - Pantalla principal en Rave

Para que el nuevo gadget se vea en mejores condiciones, crearemos una nueva página dentro de nuestro portal, pulsando en la pestaña “+” situada junto al título de la página actual “Default Home”.

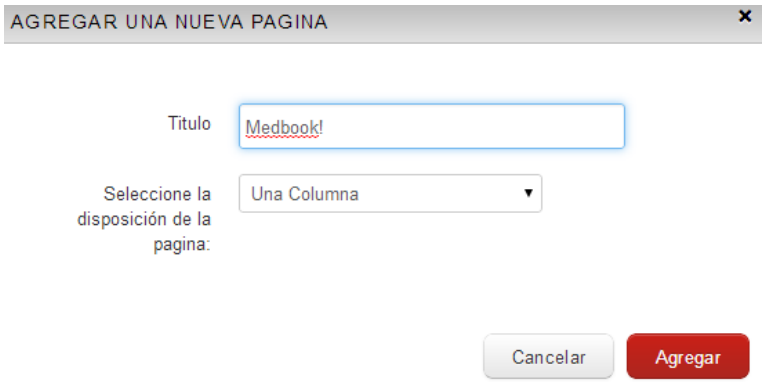


Ilustración 45 - Agregar nueva página en Rave

La nueva página se creará y aparecerá con el siguiente contenido:

Ilustración 46 - Nueva página creada

Pulsaremos en "Agregue widgets a esta página" para acceder a la *Widget store* donde podremos localizar la aplicación.

En la *Widget Store,* los gadgets salen ordenados alfabéticamente, por lo que deberemos de localizar los gadgets que empieza por "M" para encontrarlo (en la segunda página es donde se debería de localizar).

Una vez lo hemos localizado, lo agregaremos pulsando en "Agregue a la página"

Ilustración 47 - Agregando el gadget a Rave

El widget se agregará a la página y ya podremos disfrutar de nuestra aplicación.

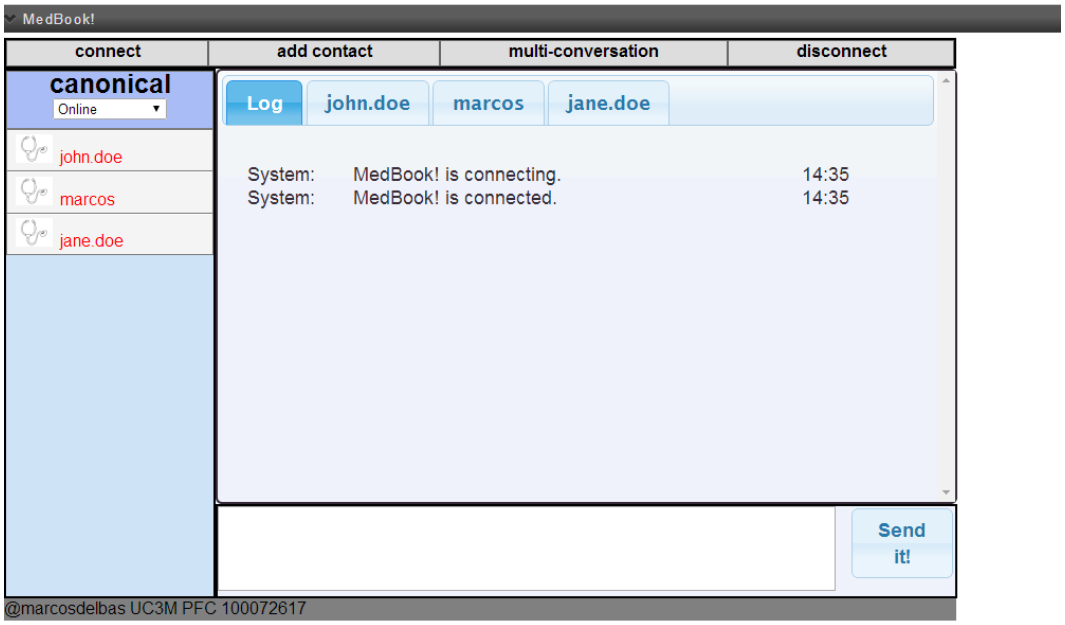


Ilustración 48 - Apariencia del gadget instalado en Rave

A.1.1 Requisitos

Los únicos requisitos de la aplicación es que sea ejecutada en un navegador que soporte HTML4.01, siendo recomendados *Mozilla Firefox* y *Google Chrome*.

A.1.2 Apache Rave

El contenedor que servirá nuestra aplicación es Apache Rave. Es conveniente que se conozcan cuáles son las funciones que disponemos dentro de esta red social. Para ello podemos echar un vistazo al punto 2 de esta memoria, aunque trataremos de resumir aquí las funciones más usadas.

Agregar widgets: Hemos explicado anteriormente el proceso para agregar nuestra aplicación a una nueva página dentro de Apache Rave. Pues bien, podemos realizar el mismo proceso para agregar otros widgets ya existentes dentro de la red social y así complementar nuestra página.

Agregar amigos: Para agregar a nuevos amigos, primero debemos acceder a nuestro perfil (botón situado en la parte superior derecha).

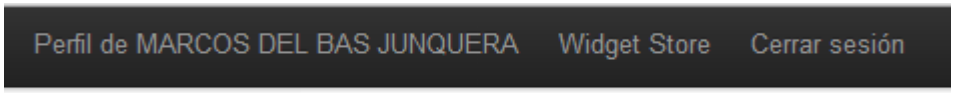


Ilustración 49 - Acceder al perfil de Rave

Una vez en nuestro perfil accederemos a la lista de amigos dentro de la red social pulsando el botón situado en la parte derecha de nuestro perfil "Agregar/eliminar amigos".

Automáticamente se desplegará un diálogo con la lista de usuarios de la red social.

Ilustración 50 - Agregar amigos a la red social

Pulsaremos en agregar en aquellos que nos interesen, y de esta manera quedará integrado dentro de nuestra lista de amigos.

A.2 Descripción del programa

En esta sección nos adentraremos en el uso del gadget *Medbook!*, en sus funciones y en las opciones sobre las mismas.

A.2.1 Visión general

Medbook! es un gadget OpenSocial creado para un entorno profesional sanitario en el que los usuarios necesiten compartir información en tiempo real.

A través de esta aplicación podremos chatear con nuestros compañeros de trabajo, crear listas de amigos, ver su estado de disponibilidad y establecer el nuestro propio para que los demás puedan vernos.

A.2.2 Funciones y servicios

Las funciones y servicios a los que nos da acceso esta aplicación tratan de resolver un problema principalmente: La comunicación instantánea desde cualquier lugar entre los distintos usuarios dentro de una empresa.

Mensajería instantánea: Este servicio nos permite comunicarnos con los compañeros de trabajo que dispongan de esta misma aplicación y a los que estemos suscritos con anterioridad.

Presencia: Este servicio nos permite anunciar nuestra presencia dentro de la aplicación y poder ver la disponibilidad de los otros usuarios desde la ventana principal de la aplicación.

Agregar Usuarios: Esta función se puede realizar de dos maneras. La más sencilla y recomendable es agregar usuarios de manera permanente desde el perfil del usuario de Apache Rave. Esta opción es permanente siempre que mantengamos a dicho usuario como amigo dentro de nuestro perfil.

La segunda opción es agregar a los usuarios desde el propio gadget. Esto nos permitirá agregar al usuario durante una sesión completa, pero una vez que cerremos la sesión este usuario desaparecerá de nuestra lista de amigos.

Aprobación de suscripción: Cuando otro usuario decide agregarnos a su lista para establecer comunicación a lo largo de una sesión debe de pedirnos permiso para acceder a nuestro estado. De esta manera, un diálogo web se desplegará preguntando por nuestra aprobación para dicha suscripción.

Eliminar Usuarios: Los usuarios se eliminar de nuestra lista siempre que los eliminemos anteriormente como amigos dentro de nuestro perfil de Apache Rave.

Conexión: El botón conexión nos permite reanudar una conexión terminada anteriormente.

Desconexión: El botón desconexión nos permite cerrar la sesión abierta en ese momento.

Anexo B: Modelo de estados de usuario con 5 estados.

B.1 Notificación de estados en un chat

B.1.1 Introducción

Muchos sistemas de mensajería instantánea incluyen notificaciones acerca del usuario con el que mantenemos una conversación one-to-one. Normalmente este tipo de notificaciones quedan limitadas a las situaciones en las que el otro usuario está tecleando. Sin embargo, este tipo de eventos elevan la sensación del usuario de estar en una sesión de chat más próxima a la realidad que si tan sólo se mostraran los mensajes.

Mientras el evento de composición del mensaje es muy interesante para este tipo de aplicaciones, el concepto de estado de sesión puede ser extendido para responder a preguntas del tipo:

- ¿Ha parado el usuario de teclear?
- ¿Está prestando realmente atención al chat?
- ¿Ha estado este usuario inactivo durante mucho tiempo?
- ¿Se ha marchado el usuario?

B.1.2 Definiciones

En esencia, las notificaciones de estados en un chat pueden entenderse como una forma de especificar la presencia del usuario en la red. La información que se intercambia puede ayudar a otros usuarios a comprender por qué no le han respondido tras diez minutos a un mensaje o por qué tras dos horas el usuario sigue sin contestarle.

Los 5 estados generales de notificación para el chat quedan descritos en la siguiente tabla:

Estado	Definición	Acciones propias del estado
<active/>	El usuario está participando actualmente en la sesión	El usuario acaba de mandar un mensaje. El usuario acaba de recibir un mensaje. El usuario presta atención a la interfaz de la aplicación o a la conversación
<inactive/>	El usuario no ha estado participando en la sesión	El usuario no ha interactuado con la interfaz durante un corto periodo de tiempo (e.g. 5 minutos)
<gone/>	El usuario ha terminado	El usuario no ha interactuado con la interfaz

	su sesión definitivamente	durante un largo periodo de tiempo (e.g. 25 minutos)
<composing/>	El usuario está componiendo un mensaje	El usuario está actualmente componiendo un mensaje (el usuario interactúa con la interfaz de la conversación).
<paused/>	El usuario ha estado componiendo un mensaje pero ha parado	El usuario ha estado componiendo un mensaje pero ha dejado de hacerlo por un corto periodo de tiempo (20 segundos)

Tabla 16 - Definición de los estados del usuario

B.1.3 Gráfico de estados

La siguiente figura pretende crear una imagen visual del modelo de 5 estados para una notificación chat.

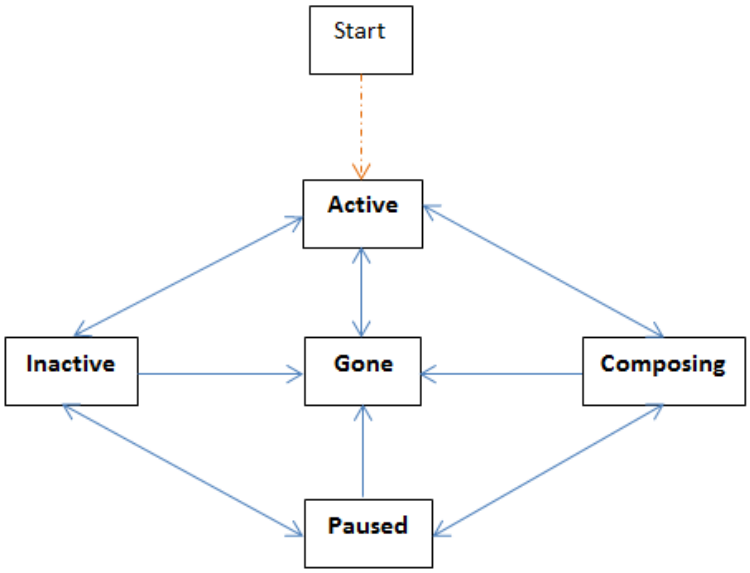


Ilustración 51 - Modelo de los 5 estados del usuario

B.2 Modelo de usuarios para Medbook!

En la siguiente lista podemos ver los 5 estados definidos para el chat.

Active: El usuario está prestando atención al chat.

Inactive: El usuario no está prestando atención al chat.

Composing: El usuario está escribiendo un mensaje.

Paused: El usuario estaba escribiendo, pero ha parado momentáneamente.

Gone: El usuario se ha ido o ha abandonado el chat.

En la siguiente imagen podemos ver un cuadro con los diferentes estados y sus transiciones.

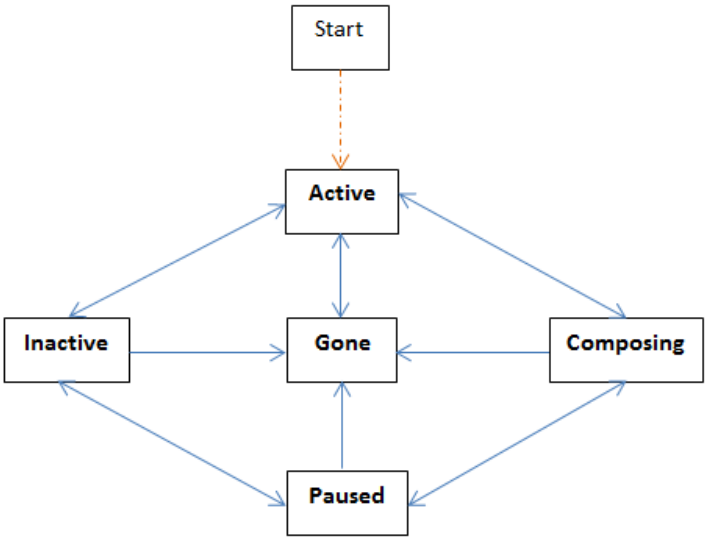


Ilustración 52 - Modelo de estados del usuario aplicable a la aplicación

Como se puede ver desde cada estado puede haber transiciones a uno o más estados. Por ejemplo, un usuario en el estado *composing* puede pasar a estar *activo* o *paused* y un usuario en el estado *gone* solamente puede pasar a *active.*

Para el caso particular de nuestra aplicación, a pesar de que un usuario puede estar en cualquier de estos 5 estados, solamente se notificará a los demás usuarios el estado cuando éste sea *active, inactive* o *gone.* Esto se ha diseñado así por simplicidad y para evitar problemas de incompatibilidad con otros clientes XMPP que puedan no tener implementados estos estados.

La situación normal es que un usuario transite entre los estados *Active-Composing-Paused-Inactive* hasta que decida abandonar la aplicación o un fallo repentino le obligue a hacerlo.