

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA DE COMPUTADORES

JAVA WEB COMMUNICATOR 2.0

Realizado por
D. David Peláez Martín
Tutorizado por
Dr. Francisco Gutiérrez López
Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, Diciembre 2016

Fecha defensa:
El Secretario del Tribunal

Resumen: El proyecto Java Web Communicator 2.0 consiste en el análisis, diseño e implementación una aplicación Web Java EE, basada en la tecnología JSF 2, que proporciona al usuario una interfaz web, que implementa un cliente de mensajería instantánea basada en el protocolo de mensajería XMPP (Extensible Messaging and Presence Protocol). Mediante la interfaz web el usuario puede establecer comunicación textual con los usuarios que tiene asociados. Asimismo, puede recibir mensajes de cualquiera de sus contactos en tiempo real. La interfaz web también permite conocer la disponibilidad de cada contacto en tiempo real. Los contactos del usuario podrán estar conectados al sistema utilizando la propia aplicación web desde otro navegador o también utilizando cualquier otra aplicación cliente XMPP. Algunos ejemplos de estas aplicaciones son: Empathy, Kopete o Jitsi. El servidor XMPP que tiene registrados los usuarios así como las relaciones entre ellos es Openfire 4.0.3 el cual sirve de fuente, tanto a la aplicación Web, como al resto de aplicaciones cliente XMPP. Para la comunicación de la aplicación Web con el servidor XMPP, se ha utilizado una librería de software libre denominada Smack 4.1.8 que proporciona una API para el uso de las funcionalidades del estándar XMPP.

Palabras claves: XMPP, Java EE, JSF 2, Openfire, Chat, Java Web Communicator.

Abstract: Java Web Communicator 2.0 project consist in the analysis, design and implementation of a Java EE Web application, based on JSF 2 technology which provides a web interface to the user that implements an instant message client based on the XMPP (Extensible Messaging and Presence Protocol).

This web interface provides to the user the possibility of stablishing textual communication with other associated contacts in real time.

The web inteface also provides real time information about the availability of every associated contact.

Contacts can be connected to the system using the web application interface or any other third party XMPP clients such Empathy, Kopete or Jitsi.

Openfire 4.0.3 is the XMPP server who manage the users and the relationship between them.

Smack 4.1.8 is the API used for the web application to communicate with the XMPP server. It provides all the XMPP standard functionalities.

Keywords: XMPP, Java EE, JSF 2, Openfire, Chat, Java Web Communicator.

Contenido

1. Introducción.....	1
1.1. Objetivos	1
1.2. Estado del arte	4
1.2.1. Mercado y protagonistas.....	4
1.2.2. Oportunidades comerciales	6
1.2.3. El boom de la mensajería instantánea en España.....	7
2. Tecnologías	9
2.1. Java.....	9
2.1.1. Disponibilidad.....	9
2.1.2. Lenguaje simple.....	9
2.1.3. Distribuido.....	9
2.1.4. Robusto	10
2.1.5. Seguro	10
2.1.6. Indiferente a la arquitectura	10
2.1.7. Portable	10
2.1.8. Alto rendimiento.....	11
2.1.9. Dinámico.....	11
2.1.10. Produce Applets y Servlets.....	11
2.2. Java EE (Java Enterprise Edition).....	11
2.3. JBoss AS.....	16
2.4. JSF 2.....	16
2.5. Push.....	21
2.6. AJAX	21
2.7. XMPP (Extensible Messaging and Presence Protocol).....	22
2.7.1. Openfire.....	23
2.7.2. Smack.....	24
2.8. Javascript	24
2.9. CSS (Cascading Style Sheets).....	24
2.10. Log4j	25
3. Arquitectura de la aplicación	27
3.1. Diagrama de clases.....	29
3.1.1. El controlador <i>LoginCtrl.class</i>	29

3.1.2.	El controlador <i>ChatCtrl.class</i>	30
3.1.3.	Características generales de los controladores	30
3.1.4.	Servicio <i>XMPPService.class</i>	31
3.2.	Casos de uso	32
3.2.1.	Autenticación	32
3.2.2.	Acceso a pantalla de chat	33
4.	Manual de instalación	37
4.1.	Guía de instalación de <i>Openfire</i>	39
4.2.	Instalación de cliente de escritorio basado en XMPP	44
5.	Conclusiones	45
6.	Líneas futuras	47
7.	Bibliografía	49

1. Introducción

Las comunicaciones en la red son de gran importancia en la actualidad y por eso, las grandes empresas de comunicación proporcionan herramientas que permiten la comunicación mediante mensajería instantánea en tiempo real en Internet para cubrir esa necesidad global.

La mensajería instantánea se identifica con las siglas IM por su traducción al inglés “*Instant Messaging*”.

Este tipo de sistema de comunicación ha existido durante décadas en diversas formas, pero recientemente ha ganado bastante popularidad debido al crecimiento de Internet y la revolución de los dispositivos móviles.

Mediante la mensajería instantánea se puede conversar con alguien que está geográficamente distante usando el ordenador.

Cuando el usuario se registra, comienza creando una lista de personas con las que se está interesado en establecer comunicación en algún momento y que ya están registradas. Cuando alguna de estas personas se autentica en el sistema, aparecerá en la lista de contactos como disponible y el usuario ya podrá iniciar una conversación con esa persona.

Normalmente se le enviarán una línea o dos y se esperará a la respuesta para volver a enviar otro mensaje corto.

Se asemeja mucho a la conversación cara a cara entre dos personas.

Una extensión de la mensajería instantánea es la introducción del concepto de grupo de conversación que consiste en que más de dos contactos comparten el mismo diálogo y cualquier de los contactos del grupo puede introducir texto en la zona común para que el resto de miembros del grupo la reciban.

Este proyecto se puede considerar una versión adaptada a las nuevas tendencias y tecnologías del proyecto fin de carrera de la Ingeniería Técnica en Informática de Sistemas que realicé en 2004.

Dicho proyecto se denominó Java Communicator e implementaba un sistema de mensajería instantánea utilizando la arquitectura cliente-servidor basada en sockets y realizada con tecnologías Java.

1.1. Objetivos

En este proyecto se desarrolla el análisis, diseño e implementación de una aplicación Web *Java EE* que proporciona al usuario una interfaz web que le permite tener acceso a un sistema

de mensajería instantánea a través de cualquier navegador Web.

La aplicación Web implementa las operaciones estándares de la mensajería instantánea utilizando el protocolo abierto estándar de mensajería instantánea *XMPP (Extensible Messaging and Presence Protocol)*. En español: Protocolo de extensible de mensajería y presencia.

Mediante el análisis, diseño e implementación de la aplicación se pretende adquirir conocimiento profundo del protocolo *XMPP* estudiando tanto como funciona el servidor *XMPP Openfire*, como la librería *Smack* que proporciona una API para operar con el servidor *XMPP*.

Por otro lado, también se pretende adquirir conocimiento en la tecnología estándar que proporciona Java para implementar aplicaciones Web. Esto es *Java Server Faces (JSF)*.

Así mismo, se pretende aprender a integrar procesos de escucha pasiva (*listeners*) en el ciclo de vida *JSF*, utilizando *Managed Beans* con el ámbito de vida requerido para cada

Además, se hace uso de tecnologías de representación de elementos en el navegador como *AJAX* utilizado mediante *JSF*, *CSS* concretamente el framework *Bootstrap* y *Javascript* para el manejo de elementos en el navegador. *JSF* utiliza gran cantidad de código *Javascript* para mantener el sincronismo entre la estructura de datos del servidor y el *DOM*.

Cabe destacar el uso de una tecnología muy potente y relativamente nueva denominada *Push*. Esta tecnología permite actualizar la pantalla del navegador del usuario de forma asíncrona disparada por una acción desde el servidor.

Diagrama del sistema implementado

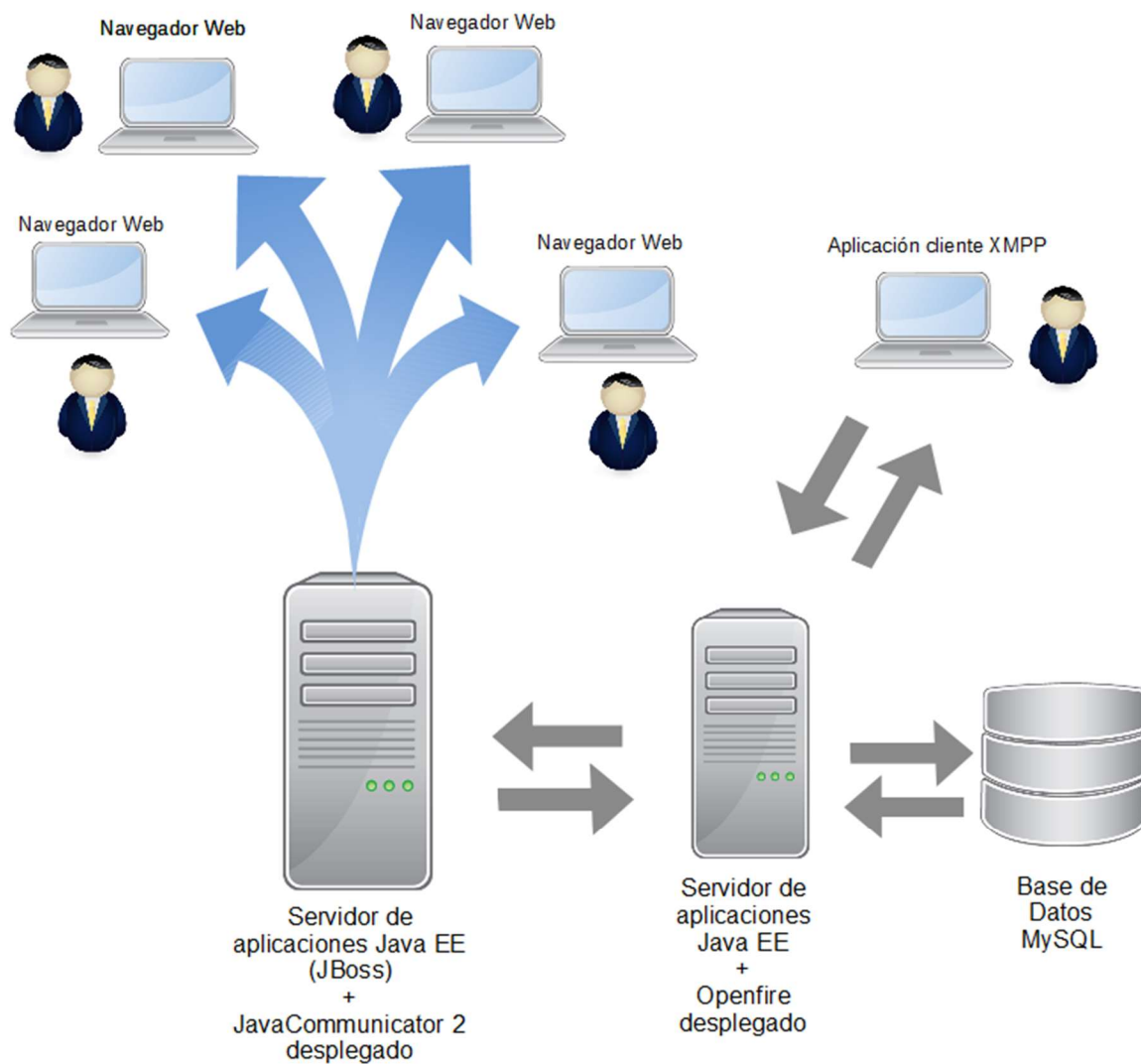


Figura 1. Diagrama de Sistema de Comunicación.

1.2. Estado del arte

Los servicios móviles de mensajería instantánea han cambiado radicalmente el modo en que las personas se comunican hoy en día. Además de su impacto social, las apps que proveen dichos servicios se han consolidado como una de las fuentes más rentables del universo móvil. El siguiente informe ofrece una perspectiva sobre el estado actual de los servicios de mensajería instantánea y las tendencias que están definiendo el desarrollo de estas apps alrededor del mundo, en particular, en España y América Latina.

1.2.1. Mercado y protagonistas

De cierta forma, los famosos 19 mil millones de dólares que *Facebook* invirtiera en la compra de *WhatsApp* ayudaron a cuantificar la importancia que hoy en día ofrece el nicho de apps de mensajería instantánea. Tal fue el impacto de dicha noticia, que la maniobra comercial del gigante de las redes sociales terminó reforzando una noticia precedente: La compra de *Viber*, por 900 millones de dólares, a manos del gigante del e-commerce japonés *Rakuten*.

PRINCIPALES APPS DE MENSAJERIA EN EL MUNDO				
App	Usuarios (millones)	Propietaria	Principales Mercados	Servicios
WhatsApp	465	Facebook	Estados Unidos, Europa, América Latina	Mensajería
Line	400	Naver Corp. (NHN)	Japón, Taiwán, Sudeste asiático	Mensajería, juegos, llamadas, stickers
WeChat	355	Tencent	China	Mensajería y servicios varios
Viber	300	Rakuten	Japón	Mensajería, llamadas, stickers
Kakao Talk	133	Kakao Corp.	Corea	Mensajería, juegos, stickers

Figura 2. Principales apps de mensajería del mundo

No es difícil entender estos movimientos corporativos si se tienen en cuenta los volúmenes asociados con las apps de mensajería instantánea. Gracias a que son gratuitas y fáciles de usar, estas aplicaciones se han convertido en una de las mayores fuentes de tráfico móvil global.

De hecho, un estudio de *Analysys Mason* reveló que más de la mitad de los usuarios de smartphones en el mundo usan de manera activa esta tipología de apps. Además de esto, el volumen de mensajes proveniente de dichas aplicaciones haya superado el volumen de mensajes de texto tradicional SMS.

De acuerdo con un estudio elaborado por *Juniper Research*, en el 2018 las apps de

mensajería instantánea representarán el 75% de todo el tráfico de mensajes móviles.

Dicha predicción señala, además, que para entonces el mercado de mensajería instantánea alcanzará un valor de 3 mil millones de dólares.

Si bien *WhatsApp* continúa siendo el principal punto de referencia con respecto a los servicios de mensajería instantánea, la competencia en dicho sector se presenta hoy en día como un juego de estrategia en donde los rivales crean alianzas, bloquean mercados y buscan constantemente aumentar su cuota de participación con la oferta de nuevos e innovadores servicios.

Para mantener su posición dominante en este juego, Jan Koum, cofundador de *WhatsApp* anunció que su famosa app incluirá durante el segundo trimestre de este año un servicio de llamadas gratuito parecido al que ofrecen varios de sus competidores. A pesar de que los amantes del concepto simplista de *WhatsApp* no hayan quedado muy contentos con dicho anuncio, este movimiento era prácticamente inevitable considerando la creciente oferta de servicios que brindan las apps de la competencia.

La expansión de la oferta de servicios de *WhatsApp* y la compra que hiciera Facebook de dicha aplicación representan una seria amenaza al margen de competitividad de otras apps. De hecho, la reciente inversión de *Facebook* en *WhatsApp* arruina los planes de *Kakao Talk* y *Line* para entrar en mercados emergentes como el de Brasil e Indonesia debido a la popularidad que goza *Facebook* en dichos países. A pesar de ello, la competencia en este sector sigue siendo ardua y todos los protagonistas del mundo de la mensajería instantánea continúan moviendo sus fichas con agresividad.

WeChat y *Line*, por ejemplo, están expandiendo su enfoque más allá de los mercados asiáticos con campañas publicitarias bastante decididas en Europa y Latinoamérica, dos mercados dominados actualmente por *WhatsApp*. De esta forma, mientras *WeChat* utiliza la imagen del famoso jugador de fútbol argentino Lionel Messi para seducir al público en América Latina, *Line* ha hecho algo parecido con distintas celebridades en medios populares de Asia, Europa y Latinoamérica.

Si bien el enfoque de *Kakao Talk* es menos agresivo en términos publicitarios, el líder de la mensajería instantánea en Corea está dispuesto a consolidar su posición en el mercado local a través del desarrollo de productos de m-commerce y juegos móviles. Por el mismo camino, Rakuten espera consolidar a *Viber* como una extensión importante de su negocio de comercio electrónico recientemente reforzado con las compras de compañías como *Kobo* y *Wuaki.tv*.

Además de los grandes colosos de la mensajería instantánea, el mercado le ha dado la bienvenida a un sinnúmero de apps innovadoras que incluyen a la canadiense *Kik*, *Telegram*, *Tango* (una app gratuita de video chat que ofrece diferentes funcionalidades como video llamadas y juegos que los usuarios pueden jugar con sus amigos mientras hablan con ellos), *Nimbuzz* (competidor fuerte de *Facebook messenger* en India en donde cuenta con 25 millones de usuarios), *Hike Messenger* y *MessageMe*.

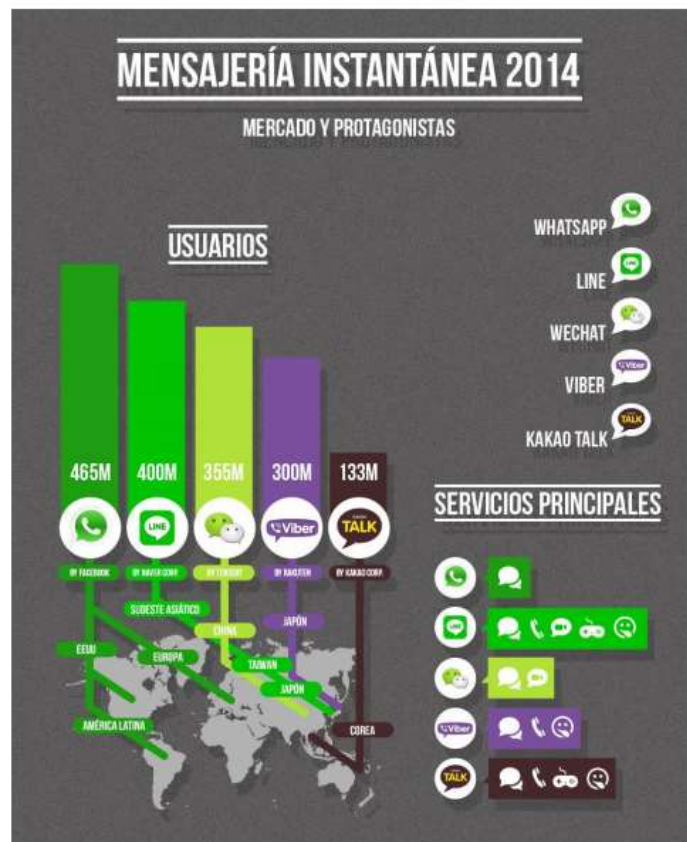


Figura 3. Mensajería instantánea 2014.

1.2.2. Oportunidades comerciales

Las apps de mensajería instantánea móvil ofrecen grandes posibilidades comerciales que van mucho más allá de los mensajes. Los usuarios de WeChat, por ejemplo, pueden reservar taxis, recargar el crédito de su teléfono y llevar a cabo operaciones complejas como invertir en productos financieros. Dichas actividades le han permitido a *Tencent*, propietaria de *WeChat*, la oportunidad de recaudar generosos ingresos por concepto de monetización de contenido móvil. De hecho, *Tencent* está enfocada en transformar *WeChat* y sus otras apps de comunicación en plataformas sociales capaces de “respaldar un amplio portafolio de aplicaciones asociadas que incluyan actividades como juegos, entretenimiento e información”.

De forma parecida, la versión tailandesa de *Line* ha lanzado con bastante éxito ventas relámpago de productos de belleza logrando acaparar el interés de millones de usuarios en dicho país. Un tipo de iniciativa que demuestra de manera clara el potencial que ofrecen las apps de mensajería instantánea como canales de distribución. Precisamente esta idea de negocios fue la que motivó a *Rakuten* en la compra de *Viber*. “Para *Rakuten* era difícil ir a algunos pequeños mercados emergentes debido a la falta de una escala o infraestructura de distribución. Sin embargo, después de la compra de *Viber* ya podemos ir a dichos mercados con nuestro negocio digital”, afirmó Hiroshi Mikitani, CEO de *Rakuten*.

1.2.3. El boom de la mensajería instantánea en España

De acuerdo con el Estudio General de Medios (*EGM*) publicado a finales del año pasado por la Asociación para la Investigación de Medios de Comunicación (*AIMC*), la mensajería instantánea ha superado al correo electrónico por primera vez en España.

Paralelo a lo anterior, y muy posiblemente debido a la popularidad que tienen las apps de mensajería instantánea en España (sobre todo *WhatsApp*), el *EGM* señala que el uso de apps subió en un 7% durante el 2013 llegando al 24% de la población.

A la par con dichas cifras, un estudio de la operadora virtual *MásMóvil* ha establecido que el consumo de Internet móvil en España ha crecido en un 20% durante el último año. “La tendencia a utilizar cada vez más Internet y con él aplicaciones de mensajería instantánea en lugar de llamar ya es una realidad”, afirma Jaume *Montané*, cofundador del *Grupo Monsan*.

Además de las ventajas inherentes que brinda la mensajería instantánea gratuita, la popularidad que estas apps tienen en España está estrechamente relacionada con los altos costes que cobran las operadoras telefónicas por llamadas y mensajes de texto SMS. Gracias a ello, España se ha convertido en el país europeo con el mayor crecimiento de mensajería instantánea.

Datos elaborados por el estudio anual del *Interactive Advertising Bureau (IAB)* refuerzan todas estas tendencias. De hecho, por primera vez este informe ha tomado en consideración la opinión de los usuarios de *WhatsApp* (88% de los internautas) descubriendo que el 59% de ellos consideran dicha app una red social.

Igualmente, el 69% de los usuarios de *WhatsApp* afirmaron que lo usan más que *Facebook*, la red social más popular en España de acuerdo con dicho estudio.

Tomando en cuenta todas estas tendencias y la compra que hiciera *Facebook* de *WhatsApp*, es indudable que la navegación móvil en España estará bastante relacionada con estas dos populares marcas del mundo virtual. En todo caso, será interesante seguir el desarrollo de los servicios de mensajería instantánea en un país que como España goza de un liderazgo importante en el desarrollo de apps.

2. Tecnologías

2.1. Java

Es un lenguaje de programación que se engloba dentro del paradigma orientado a objetos.

En la última versión de Java, la Java 8 se introduce el concepto de funciones lambda que hace que Java pase a englobarse también en los lenguajes de programación funcionales.

A diferencia de otros lenguajes de programación, Java compila el código en un metalenguaje denominado *Bytecode* y requiere ser ejecutada en una JVM (Máquina Virtual Java) que es la propia de la arquitectura del sistema anfitrión y traduce el *Bytecode* a código máquina.

Aunque no comenzó como un proyecto enfocado a Internet, Oracle ha sabido evolucionar para dar soporte a los requerimientos de las aplicaciones actuales.

Todos los conceptos en los que se apoya el paradigma orientado a objetos, encapsulación, herencia, polimorfismo, etc., están presentes en Java.

2.1.1. Disponibilidad

De un amplio conjunto de bibliotecas. Como ya se mencionó anteriormente, Java es algo más que un lenguaje. La programación de aplicaciones con Java se basa no solo en el empleo del juego de instrucciones que componen el lenguaje, sino, fundamentalmente, en la posibilidad de utilizar el amplísimo conjunto de clases que Oracle pone a disposición del programador y con las cuales es posible realizar prácticamente cualquier tipo de aplicación.

2.1.2. Lenguaje simple

Java posee una curva de aprendizaje muy rápida. Resulta relativamente sencillo escribir *Applets* interesantes desde el principio. Todos aquellos familiarizados con C++ encontrarán que Java es más sencillo, ya que se han eliminado ciertas características, como los punteros. Debido a su semejanza con C y C++, y dado que la mayoría de la gente los conoce, aunque sea de forma elemental, resulta muy fácil aprender Java. Los programadores experimentados en C++ pueden migrar muy rápidamente a Java y ser productivos en poco tiempo.

2.1.3. Distribuido

Java proporciona una colección de clases para su uso en aplicaciones de red, que permiten abrir sockets y establecer y aceptar conexiones con servidores o clientes remotos, facilitando así la creación de aplicaciones distribuidas.

Interpretado y compilado a la vez. Java es compilado, en la medida en que su código fuente se transforma en una especie de código máquina, el *Bytecode*, semejantes a las instrucciones de ensamblador. Por otra parte, es interpretado, ya que el *Bytecode* se pueden ejecutar directamente sobre cualquier máquina a la cual se hayan portado el intérprete y el sistema de

ejecución en tiempo real (run-time).

2.1.4. Robusto

Java fue diseñado para crear software altamente fiable. Para ello proporciona numerosas comprobaciones en compilación y en tiempo de ejecución. Sus características de memoria liberan a los programadores de una familia entera de errores (la aritmética de punteros), ya que se ha prescindido por completo de los punteros, y la recolección de basura elimina la necesidad de liberación explícita de memoria.

2.1.5. Seguro

Java gestiona los errores y otros eventos, que se producen en tiempo de ejecución, mediante excepciones. En la programación siempre se producen errores, más o menos graves, pero que hay que gestionar y tratar correctamente. Por ello en Java proporciona un mecanismo consistente en el uso de bloques *try/catch/finally*. La técnica básica consiste en colocar las instrucciones que podrían provocar problemas dentro de un bloque *try*, y colocar a continuación uno o más bloques *catch*, de tal forma que si se provoca un error de un determinado tipo, lo que haremos será saltar al bloque *catch* capaz de gestionar ese tipo de error específico. El bloque *catch* contendrá el código necesario para gestionar ese tipo específico de error. Suponiendo que no se hubiesen provocado errores en el bloque *try*, nunca se ejecutarían los bloques *catch*. Si en la implementación de un método no se desea controlar los errores pero sí requerir al método que invoca a este método dicho control, se etiqueta el método donde se producen los errores con la palabra *throws* seguido del conjunto de excepciones que deberá controlar en el método llamante.

Java proporciona una metodología muy potente para controlar errores en tiempo de ejecución.

2.1.6. Indiferente a la arquitectura

Java está diseñado para soportar aplicaciones que serán ejecutadas en los más variados entornos de red, desde Unix a *Windows*, pasando por *Mac* y estaciones de trabajo, sobre arquitecturas distintas y con sistemas operativos diversos. Para acomodar requisitos de ejecución tan diversos o variopintos, el compilador de Java genera *ByteCode*: un formato intermedio indiferente a la arquitectura diseñado para transportar el código eficientemente a múltiples plataformas hardware y software. El resto de problemas los soluciona el intérprete de Java.

2.1.7. Portable

La indiferencia a la arquitectura representa sólo una parte de su portabilidad. Además, Java especifica los tamaños de sus tipos de datos básicos y el comportamiento de sus operadores aritméticos, de manera que los programas son iguales en todas las plataformas. Estas dos últimas características se conocen como la *Máquina Virtual Java* (JVM).

2.1.8. Alto rendimiento

Multihebra. Hoy en día ya se ven como terriblemente limitadas las aplicaciones que sólo pueden ejecutar una acción a la vez. Java soporta sincronización de múltiples hilos de ejecución (*multithreading*) a nivel de lenguaje, especialmente útiles en la creación de aplicaciones de red distribuidas. Así, mientras un hilo se encarga de la comunicación, otro puede interactuar con el usuario mientras otro presenta una animación en pantalla y otro realiza cálculos.

2.1.9. Dinámico

El lenguaje Java y su sistema de ejecución en tiempo real son dinámicos en la fase de enlazado. Las clases sólo se enlazan a medida que son necesitadas. Se pueden enlazar nuevos módulos de código bajo demanda, procedente de fuentes muy variadas, incluso desde la Red.

2.1.10. Produce Applets y Servlets

Java puede ser usado para crear tres tipos de programas: aplicaciones independientes, *Applets* y *Servlets*. Las aplicaciones independientes se comportan como cualquier otro programa escrito en cualquier lenguaje, como por ejemplo el navegador de Web *HotJava*, escrito íntegramente en *Java*. Por su parte, los *Applets* son pequeños programas que aparecen embebidos en las páginas Web, como aparecen los gráficos o el texto, pero con la capacidad de ejecutar acciones muy complejas, como animar imágenes, establecer conexiones de red, presentar menús y cuadros de diálogo para luego emprender acciones, etc. Finalmente, los *Servlets*, que son clases que proporcionan un servicio a través del protocolo *HTTP*. Los *Servlets* reciben peticiones desde un navegador Web, las procesan y devuelven una respuesta al navegador, normalmente *HTML*. Para realizar estas tareas pueden utilizar cualesquiera clases incluida en el lenguaje Java. Los *Servlets* requieren ser ejecutados en un *Contenedor de Servlets*.

2.2. Java EE (Java Enterprise Edition)

Hace ya unos años, en 2013, fue publicada una nueva versión del conjunto de especificaciones que forman parte de *Java EE* con varias novedades y mejoras que aumentan la productividad. *Java EE 7* es el conjunto de especificaciones a disposición de las aplicaciones empresariales que son implementadas por cada servidor de aplicaciones donde se ejecutan las aplicaciones Java, estas especificaciones describen la funcionalidad y permiten que una aplicación pueda ser ejecutada en cualquier servidor de aplicaciones Java que las implemente, ya sea *WildFly*, *WebLogic* o *WebSphere*, sin necesidad de grandes cambios o ninguno en absoluto al proporcionar interoperabilidad entre diferentes implementaciones con el objetivo de no estar encadenado a un determinado vendedor.

Java EE usa como lenguaje de programación Java que incorporando numerosas novedades en la versión 8 lo hacen seguir siendo uno de los mejores lenguajes y más usado para el

desarrollo de aplicaciones empresariales. Estos cambios proporcionan muchas mejoras en el lenguaje Java en general.

El modelo clásico de capas en la arquitectura de las aplicaciones *Java EE* se divide en las siguientes:

Cliente: normalmente se trata de un navegador, pero puede ser un teléfono inteligente o una computadora de escritorio, incluso otra aplicación. Es la que presenta la información al usuario.

Capa web: se comunica con el cliente y la capa de negocio. Obtiene los datos y los transforma al formato adecuado al cliente generalmente HTML o JSON.

Capa de negocio: proporciona y persiste los datos de la capa cliente y contiene la lógica de negocio de la aplicación. Se ejecuta en un servidor de aplicaciones o contenedor de *Servlets*.

Sistemas de información: donde se persisten los datos de la aplicación, puede ser una base de datos relacional como Oracle, *MySQL* o *PostgreSQL* o una base de datos *NoSQL* como *Redis* o *MongoDB* u otros sistemas como *Elasticsearch*.

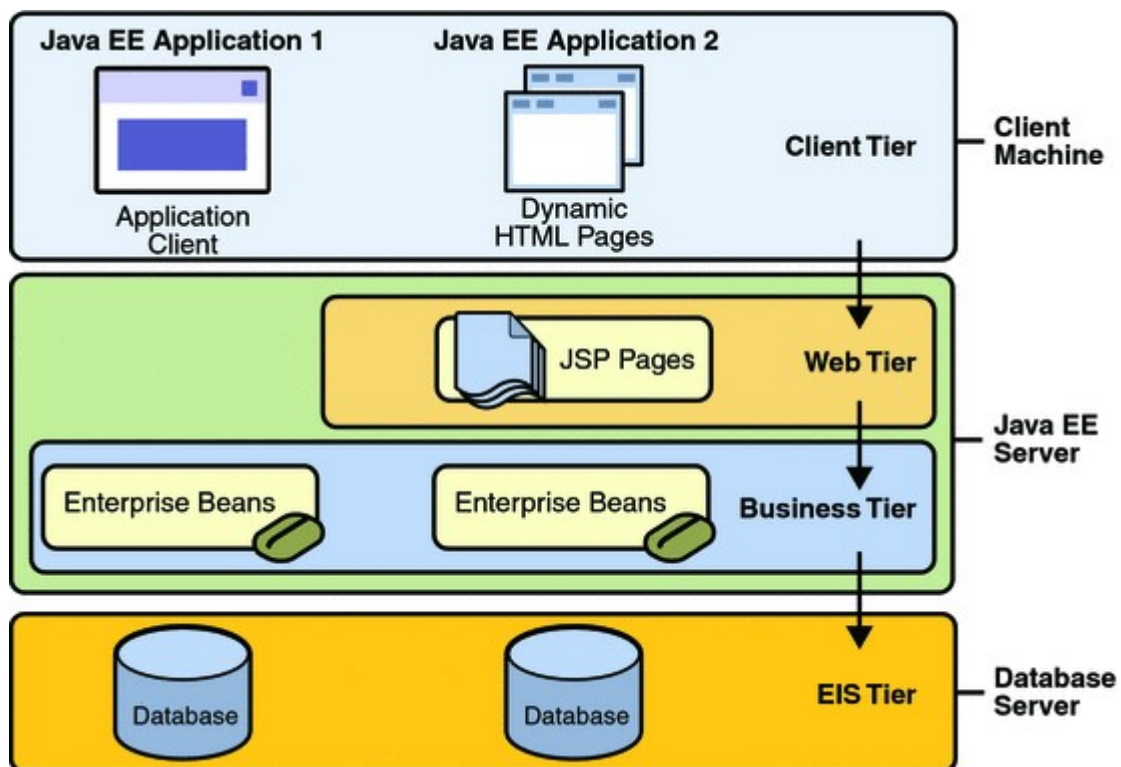


Figura 4. Aplicaciones multicapa Java EE

En el listado de especificaciones encontramos algunas dedicadas a persistencia en base de datos (*JDBC*, *JPA*), transaccionalidad (*JTA*), procesamiento de peticiones *HTTP* (*Servlets*, *JSF*, *REST*), generación de *HTML* (*Servlets*, *JSP*, *JSF*), servicios web

basados en *REST* y *SOAP* (*JAX-RS*, *JAX-WS*), soporte para websockets en el lado del servidor y cliente, tratamiento de JSON, validación de objetos, comunicación entre aplicaciones desacoplada con mensajes (*JMS*), concurrencia, servicios de nombres y descubrimiento o trabajos en lotes entre otras.

Las especificaciones y versiones que componen *Java EE 7* son:

Tecnologías aplicaciones Web (websockets, html)

- Java API for *WebSocket* 1.0 (nueva)
- Java API for *JSON Processing* 1.0 (nueva)
- Java *Servlet* 3.1
- *JavaServer Faces* (*JSF*) 2.2
- *Expression Language* (*EL*) 3.0
- *JavaServer Pages* (*JSP*) 2.3

Tecnologías aplicaciones empresariales (persistencia, transaccionalidad, inyección de dependencias, concurrencia, validación, mensajería, correos electrónicos)

- Aplicaciones Batch para *Java Platform* 1.0 (nueva)
- Utilidades para concurrencia para *Java EE* 1.0 (nueva)
- Contextos e Inyección de dependencias para Java (*CDI*) 1.1
- Inyección de dependencias para Java 1.0
- Persistencia Java (*JPA*) 2.1
- API para transacciones Java (*JTA*) 1.2
- API de Servicio de Mensajes (*JMS*) 2.0
- Enterprise JavaBeans (*EJB*) 3.2
- Bean Validation 1.1
- JavaMail 1.5
- Interceptors 1.2
- *Java EE* Connector Architecture (*JCA*) 1.7
- Common Annotations for the Java Platform 1.2
- Tecnologías servicios web (servicios web *REST*, *SOAP*)
- Java API for RESTful Web Services (*JAX-RS*) 2.0
- Java API for XML-Based Web Services (*JAX-WS*) 2.2
- Implementing Enterprise Web Services 1.3
- Metadatos de servicios web para la plataforma Java
- Java API for XML-Based RPC (*JAX-RPC*) 1.1 (Opcional)
- Java API for XML Messaging 1.3
- Java API for XML Registries (*JAXR*) 1.0

Tecnologías de gestión y seguridad (autenticación, autorización)

- Java Authentication Service Provider Interface for Containers (JASPIC) 1.1
- Java Authorization Contract for Containers (JACC) 1.5
- *Java EE* Application Deployment 1.2 (Opcional)
- J2EE Management 1.1
- Soporte para Debug para otros lenguajes 1.0

Especificaciones relacionadas con *Java EE* en Java SE (bases de datos, XML, gestión)

- Java Database Connectivity (JDBC) 4.0
- Java Architecture for XML Binding (JAXB) 2.2
- Java Management Extensions (JMX) 2.0
- JavaBeans Activation Framework (JAF) 1.1
- Java API for XML Processing (JAXP) 1.3
- Streaming API for XML (StAX) 1.0

Además de incorporar nuevas especificaciones y las existentes recibir mejoras entre las novedades se encuentran varias que facilitan el desarrollo posibilitando en gran medida prescindir de configuración en archivos XML propensos a errores, ya se inició en *Java EE 6*, sustituyéndose por definiciones declarativas con anotaciones. Entre las mejoras están:

- Usando la nueva funcionalidad de entrada y salida (NIO) de Java SE los *Servlets* pueden manejar comunicación asíncrona. Se permite upgrade del protocolo lo que permite por ejemplo empezar una petición como *HTTP/1.1* y pasar a usar *HTTP/2*, en WildFly esto es usado para reducir el número de puertos abiertos.
- *JPA* ahora puede invocar procedimientos almacenados, ejecutar sentencias *SQL* de update y delete masivas y controlar que entidades son cargadas de forma ansiosa (eager) o vaga (lazy).
- Se continúa avanzando en lo iniciado en versiones anteriores permitiendo usar *POJO* con anotaciones para facilitar el desarrollo.
- *JTA* define una nueva anotación que permite a cualquier bean *CDI* usar transacciones.
- En JAX-RS se añade una API cliente para invocar endpoints *REST*, se añade soporte para E/S asíncrona tanto para el cliente como para el servidor y hypermedia linking.
- Bean Validation permite validación a nivel de método con mejor integración en el resto de la plataforma *Java EE*.

Los servidores de aplicaciones Java pueden implementar todas las especificaciones

denominándose *full-profile* como *JBoss/WildFly*, *WebLogic* o *WebSphere*. Sin embargo, algunas aplicaciones no necesitan todas las funcionalidades definidas en *Java EE* por lo que algunos servidores como *TomEE* también *WildFly* pueden implementar únicamente un subconjunto para la generación de contenido web, denominándose así *web-profile*. Otros servidores como *Tomcat* y *Jetty* son contenedores de servlets que soportan un grupo más reducido de especificaciones de las tecnologías web (únicamente *Servlet*, *JSP*, *EL* y *WebSocket*) pero que siguen siendo suficientes para algunas aplicaciones o usando algunas equivalentes proporcionadas por Spring.

Tecnologías *web-profile*

- *Java Servlet API*
- *Enterprise Java Bean Lite*
- Context and Dependency Injection
- *Java Server Faces*
- *Java Transaction API*
- *Java Persistence API*
- *Web Socket*
- *Bean Validation*
- *JAX-RS*
- *JSON-P*

Java EE 7 fue publicada hace ya varios años y muy posiblemente muchas entidades públicas y empresas privadas seguirán usando versiones más antiguas para sus aplicaciones. En el mundo empresarial las aplicaciones se han de mantener funcionando algunos lustros o décadas, para atender este requisito Java en sus 20 años se ha caracterizado por ofrecer compatibilidad hacia atrás en cada nueva versión e incorporar en el lenguaje solo aquellas mejoras que se han mostrado útiles. Esto puede hacer parecer que avanza lentamente, al menos más que otras tecnologías, y que usando las versiones anteriores hoy parezcan obsoletas, en su momento XML era uno de los formatos más empleados para realizar configuración y se usaba profusamente, hoy se están prefiriendo formatos menos verbosos y legibles como JSON y YAML. Java 8 ofrece varias novedades en el lenguaje y *Java EE 7* usando anotaciones en gran medida hace innecesarios los ficheros XML ambos siguiendo las nuevas tendencias del desarrollo y programación. Algunas personas comparan las tecnologías que prefieren con lo que recuerdan de versiones antiguas de Java o *Java EE*.

Java EE ofrece a los desarrolladores un conjunto de especificaciones que cubren las necesidades de un gran número de aplicaciones empresariales y la plataforma ofrece garantías a largo plazo como ha demostrado que le hacen seguir siendo una de

opciones preferidas para el desarrollo. La alternativa a *Java EE* más usada es Spring y sus numerosos proyectos que proporcionan funcionalidades similares sin necesidad de un contenedor *full-profile*. En cualquiera de estas dos opciones la base de procesamiento y generación de *HTML* en el lado del servidor son los *servlets* pero estos trabajan a bajo nivel, no se suelen usar directamente prefiriéndose frameworks de más alto nivel como Spring MVC, Apache Tapestry, Grails, Struts u otros. Del mismo modo para el acceso a la base de datos está *JDBC* pero tampoco se suele usar directamente por ofrecer una API a bajo nivel prefiriendo *Hibernate*, *JPA* o *jOOQ* como alternativa a *JDBC* a los ORM anteriores.

Bibliografía recomendada para un conocimiento más profundo sobre *Java EE 7* son: *Java EE 7*, *Java EE 7 Development with WildFly*, *Java EE 7 Developer Handbook*, finalmente también es posible consultar el tutorial oficial de *Java EE 7*.

El futuro *Java EE 8* está planificado para 2017 fecha también planificada para *Java 9* en la que se añadirá soporte para *HTTP/2* y será la versión 4.0 de los *Servlets*. Pero no hace falta esperar hasta el 2017 para aprovecharnos hoy de las ventajas de *HTTP/2* tanto para clientes como servidores, los principales navegadores ya lo soportan y se puede configurar *HTTP/2* en varios servidores web y de aplicaciones Java como Nginx, Apache *HTTPD*, *WildFly* o *Jetty*. También se ha anunciado que como alternativa a *JSF* basado en componentes se proporcionará una especificación que implemente el patrón MVC basado en acciones.

2.3. JBoss AS

Es un servidor de aplicaciones *Java EE* de código abierto implementado en Java puro. Al estar implementado en Java puede ser ejecutado en cualquier sistema operativo con el único requisito de que tenga la Máquina Virtual Java.

Se estructura en distintos subsistemas que se activan en función de las necesidades de los componentes desplegados. De esta manera, implementa todas las funcionalidades exigidas para considerarse Servidor de Aplicaciones Java y no por ello tiene un tiempo de carga alto ni consume excesiva memoria en su ejecución.

2.4. JSF 2

JavaServer Faces es el framework oficial de Java Enterprise para el desarrollo de interfaces de usuario avanzadas en aplicaciones web. La especificación de *JSF* ha ido evolucionando desde su lanzamiento en 2004 y se ha ido consolidando, introduciendo nuevas características y funcionalidades.

La especificación original de *JavaServer Faces* (1.0) se aprobó en marzo del 2004, con la Java Specification *Request JSR 127*. En esta especificación se define el

funcionamiento básico de *JSF*, introduciéndose sus características principales: uso de beans gestionados el lenguaje *JSF EL*, componentes básicos y navegación entre vistas.

La especificación *JavaServer Faces* 1.2 se aprobó en mayo del 2006. La JSR donde se define es la JSR 252. En esta especificación se introducen algunas mejoras y correcciones en la especificación 1.1. Una de las principales es la introducción del lenguaje unificado de expresiones (*Unified Expression Language* o *Unified EL*), que integra el lenguaje *JSTL* de expresiones de *JSP* y el lenguaje de expresiones de *JSF* en una única especificación.

La especificación actual de *JSF* (*Java Server Faces* 2.2) se aprobó en mayo de 2013 (JSR 344) y la especificación de *Java EE 7* se aprobó en 2015. En la especificación de la versión de *JSF* 2.1 se rompió definitivamente con *JSP* como forma de definir las vistas *JSF* y se introdujo un formato independiente de *JSP*. De hecho, la implementación de referencia de Oracle se integra con *Facelets*, un sistema de definición de plantillas para las páginas web que sustituye a *JSP* y que se ha popularizado con *JSF* 2.1.

Algunas características importantes de esta especificación son:

Soporte para *AJAX*.

Componentes múltiples.

Integración con *Facelets*.

Gestión de recursos (hojas de estilo, imágenes, etc.)

Facilidad de desarrollo y despliegue.

JSF surge para dar soporte a las aplicaciones *RIA* (*Rich Internet Applications*) que son aplicaciones que ofrece una experiencia al usuario similar a la que ofrecen las aplicaciones de escritorio, como pueden ser: marcar posiciones en un mapa, imprimir, enviar ficheros, etc.

JSF se basa en *AJAX* para ofrecer esa flexibilidad en las funcionalidades.

La definición de la interfaz en *JSF* 2 se realiza en forma de páginas *XHTML* con distintos tipos de etiquetas. Estas páginas se denominan páginas *JSF*. La siguiente figura muestra el funcionamiento de *JSF* para generar una página por primera vez.

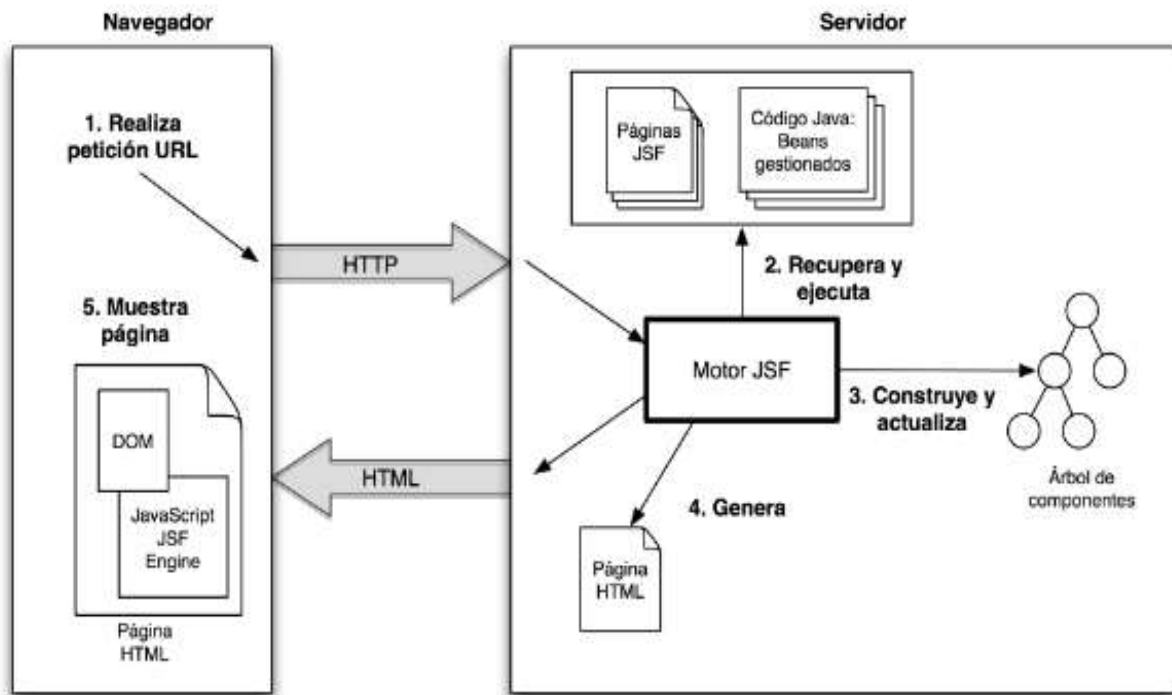


Figura 5. Interacción del servidor con el navegador.

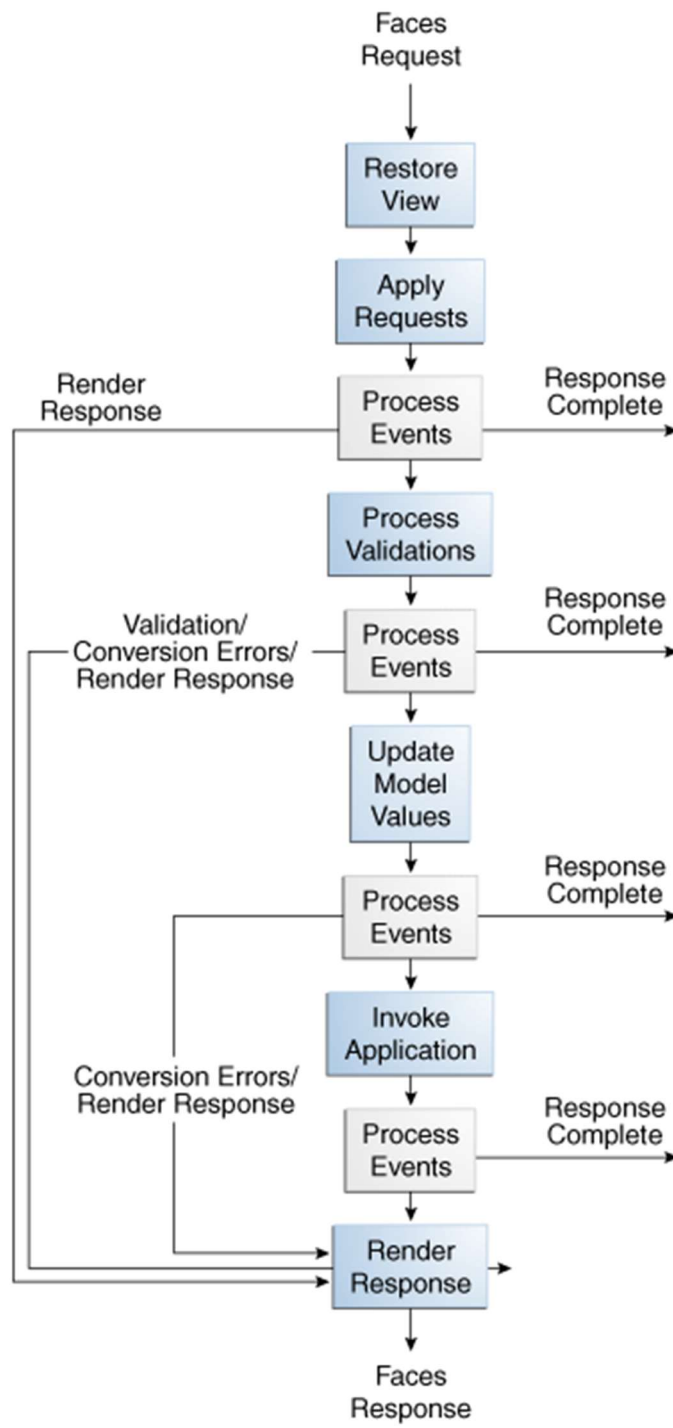


Figura 6. Ciclo de vida en fases JSF.

En el ciclo de vida de procesamiento de peticiones *JSF* se realizan todas las tareas de back-end que de otro modo debería realizar el programador. Procesa los parámetros de entrada y gestiona un conjunto de componentes del lado del servidor y los sincroniza para que lo vea el usuario en el navegador.

Realiza la mayoría de las operaciones del lado del servidor de una manera automática basada en eventos. Enlaza directamente campos con propiedades de una clase Java, de modo que sincroniza *beans* del lado del servidor, con los componentes de UI que ve el usuario. Es lo que se conoce como “*gestión de estado*” y es la piedra angular de *JSF*.

No se almacena el estado, esto es, una transacción entre el cliente y el servidor, no almacena en memoria las transacciones anteriores. *JSF* mantiene una vista en el lado del servidor que representa partes importantes del estado actual del cliente. Las fases del ciclo de vida son:

- **Crea o restaura la vista** → Crea o restaura un árbol de componentes (vista) del lado del servidor, que representan los componentes de interfaz de usuario del cliente. Las vistas se crean y almacenan en un contenedor de vistas llamado “*FacesContext*”. El desarrollador no se tiene que preocupar por si los datos se mezclan al realizar múltiples “*request*” ya que cada petición irá en su propio hilo.
- **Aplica los valores del “*request*”** → Actualiza los componentes del lado del servidor con datos “frescos” que provienen del lado del cliente. Esto se hace a través de una invocación a “*processDecodes()*” en el árbol de componentes (*UIViewRoot*), lo que provocará que se llame de manera recursiva al método “*processDecodes()*” de cada uno de los componentes hijo y los métodos “*decode()*” de los componentes UI. Los componentes UI que tienen el atributo “*value*”, como los campos de texto, etc., implementarán el interfaz “*ValueHolder*”, los componentes de formulario que tienen valores editables por el usuario, implementan “*EditableValueHolder*” y finalmente aquellos componentes que ejecutan acciones, como los botones o los links, implementarán “*ActionSource*”.
- **Ejecuta las Validaciones** → Procesa las validaciones y realiza la conversión de datos en su caso. Para hacer esto invoca al método “*processValidators()*” de manera recursiva en el árbol de componentes (*UIViewRoot*). Cualquier componente que falle en la validación o en la conversión tendrá la propiedad “*valid = false*”
- **Actualiza el modelo** → Una vez que se pasan las validaciones y las conversiones, se realizan las actualizaciones de los objetos que modelan los datos en el lado del servidor (*Managed Beans*). El mecanismo es similar a los anteriores, se invoca al interfaz “*processUpdates()*” de manera recursiva en el árbol de componentes *UIViewRoot*

- **Invoca a la lógica de aplicación** → En esta fase se invoca a los “action method”, que son métodos previamente implementados, que realizan las acciones necesarias con los datos. Se invoca al método “*processApplication()*” del *UIViewRoot* y se transmite eventos en cola a cada componente que implemente “*ActionSource*”.
- **Devuelve la respuesta** → Salva el estado y devuelve la respuesta al cliente para que los componentes sean representados. El mecanismo es similar al que hemos comentado anteriormente, se invocará en cascada a los métodos “*encodeXX()*” de cada componente. El lenguaje de “renderizado” puede ser cualquiera, html, xml, etc. Se almacena el estado de la vista para que pueda ser utilizado en las siguientes peticiones.

2.5. Push

La tecnología *Push* o *Server Push*, describe un estilo de comunicación basado en Internet donde la petición para una transacción es inicializada por el publicador o el servidor central. Es la técnica contraria a *Pull/get*, donde la petición para la transmisión de la información es inicializada por el receptor o cliente.

Los servicios de *Push* están normalmente basados en las preferencias de la información. Este modelo se denomina publicador/suscriptor. Un cliente “se suscribe” a canales de información proporcionados por el servidor. Cuando hay información disponible en alguno de estos canales, el servidor publica la información en el canal y, de esta forma, llega a los clientes suscritos al correspondiente canal.

Hay diversas implementaciones de esta tecnología: *WebPush*, *HTTP server Push*, *Pushlet*, *Long polling*, *Flash XMLSockets relays*...

Para este proyecto se ha utilizado la API *WebSockets* de forma indirecta pues, está implícita en la librería *JSF Omnifaces* que se ha utilizado para implementar esta funcionalidad.

2.6. AJAX

AJAX, acrónimo de *Asynchronous Javascript And XML* (*Javascript* asíncrono y *XML*), es una técnica de desarrollo web para crear aplicaciones interactivas o *RIA* (*Rich Internet Applications*). Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, mejorando la interactividad, velocidad y

usabilidad en las aplicaciones.

AJAX es una tecnología asíncrona, en el sentido de que los datos adicionales se solicitan al servidor y se cargan en segundo plano sin interferir con la visualización ni el comportamiento de la página, aunque existe la posibilidad de configurar las peticiones como síncronas de tal forma que la interactividad de la página se detiene hasta la espera de la respuesta por parte del servidor.

Javascript es un lenguaje de programación (scripting language) en el que normalmente se efectúan las funciones de llamada de *AJAX* mientras que el acceso a los datos se realiza mediante *XMLHttpRequest*, objeto disponible en los navegadores actuales. En cualquier caso, no es necesario que el contenido asíncrono esté formateado en XML.

AJAX es una técnica válida para múltiples plataformas y utilizable en muchos sistemas operativos y navegadores dado que está basado en estándares abiertos como *Javascript* y Document Object Model (DOM).

En este proyecto hace uso implícito de *AJAX* mediante la utilización de *JSF 2*.

2.7. *XMPP* (Extensible Messaging and Presence Protocol)

Es un protocolo abierto de mensajería instantánea que, originalmente fue desarrollado por la comunidad de software libre Jabber para proporcionar una solución libre, descentralizada y alternativa a las soluciones de mensajería instantánea cerradas. *XMPP* proporciona varias ventajas:

Abierto: El protocolo *XMPP* es libre, abierto, público y fácilmente comprensible; Así mismo existen numerosas implementaciones de servidores, clientes, componentes de servidor y librerías de código que proporcionan API's.

Estándar: La Internet Engineering Task Force (IETF) ha formalizado el núcleo de los protocolos de los flujos de información XML y ha aprobado la tecnología de mensajería instantánea y presencia.

La especificación *XMPP* se publicaron en el RFC 3920 y RFC 3921 en el año 2004.

La *XMPP Standard Foundation* sigue publicando muchos protocolos de extensiones de *XMPP*. En 2011 los núcleos *RFC*'s se revisaron y se publicaron actualizaciones que dieron lugar a las especificaciones RFC 6120, RFC 6121 y RFC 6122).

Probado: La primera implementación de la tecnología Jabber/*XMPP* fue desarrollada por Jeremie Miller in 1998 y ahora es muy estable. Cientos de desarrolladores trabajan

en estas tecnologías y hay decenas de miles de servidores *XMPP* funcionando actualmente en Internet y hay millones de usuarios utilizan mensajería instantánea *XMPP* a través de servicios públicos.

Descentralizada: La arquitectura de la red *XMPP* es similar a la de correo electrónico; como resultado, cualquiera puede ejecutar su propio servidor *XMPP*, habilitando a personas y a organizaciones tener el control de la gestión de sus comunicaciones.

Seguro: cualquier servidor *XMPP* puede estar aislado de la red pública (por ejemplo: para uso en Intranets) y, además, la especificación *XMPP* define protocolos de robustos de seguridad basados en SASL y TLS. La comunidad sigue trabajando de forma muy activa para conseguir mensajería encriptada punto a punto.

Extensible: Utilizando la potencia de XML, cualquiera puede personalizar el funcionamiento desarrollado sobre el protocolo núcleo.

Flexibilidad: Las aplicaciones *XMPP* van más allá de la mensajería instantánea. Otras aplicaciones son: sindicación a contenidos, mantenimiento de redes, herramientas colaborativas, compartición de ficheros, juegos, monitorización de sistemas remotos, servicios web, capas middleware ligeras, computación en la nube, entre otras.

2.7.1. [Openfire](#)

Es un servidor de colaboración en tiempo real (RTC) con licencia Open Source Apache desarrollado en Java. Utiliza el protocolo *XMPP* que está ampliamente aceptado como estándar de mensajería instantánea.

Openfire implementa las siguientes características:

- Panel de administración web.
- Interfaz para agregar plugins.
- SSL/TLS.
- Amigable.
- Adaptable según las necesidades.
- Conferencias.
- Interacción con MSN, Google Talk, Yahoo messenger, AIM, ICQ, Jingle.
- Estadísticas del Servidor, mensajes, paquetes, etc.
- Cluster con múltiples servidores.
- Transferencia de Archivos.
- Compresión de datos.
- Tarjetas personales con Avatar.
- Mensajes offline.

- Favoritos.
- Autenticación vía Certificados, Kerberos, LDAP, PAM y Radius.
- Almacenamiento en Active Directory, LDAP, MS SQL, MySQL, Oracle y PostgreSQL.
- SASL: ANONYMOUS, DIGEST-MD5 y Plain.

2.7.2. Smack

Es una librería de software libre, multiplataforma, fácil de usar que implementa la API con las funciones de un cliente *XMPP*.

Smack es una librería implementada completamente en Java y puede ser incluida en cualquier aplicación para implementar desde una simple integración con *XMPP* que envíe mensajes de notificación y detectores de disponibilidad e dispositivos hasta un cliente completo *XMPP*.

En este proyecto se ha utilizado la versión 4.1.8.

Cabe mencionar que, de la versión 3 a la versión 4, hubo un cambio considerable en la arquitectura de la librería que se basó en la utilización de la clase *ThreadLocal* para almacenar los manejadores de los distintos componentes como el gestor de Chat, el gestor de Presencia de los contactos, etc.

2.8. Javascript

En términos generales, *Javascript* es un lenguaje interpretado en el navegador web, aunque hay librerías que se ejecutan en el servidor como Node.js.

El funcionamiento de la aplicación tiene como requisito que el cliente tenga habilitado *Javascript* ya que, de forma implícita, *JSF* utiliza *Javascript* para realizar las llamadas *AJAX* al servidor y para la renderización parcial de la pantalla.

También se ha utilizado de forma directa para evitar la incompatibilidad de los eventos *Javascript* originados por el framework Bootstrap y los eventos generados por *JSF*.

2.9. CSS (Cascading Style Sheets)

Es un lenguaje de estilos utilizado para describir la presentación de un documento escrito en Markup Language (*XML*, *LaTeX*, *troff*) y su uso es más habitual en páginas *XHTML* y *HTML*.

CSS ha evolucionado muchísimo desde su primera versión. La que más impacto tubo fue la versión 3, *CSS3* y posteriormente se ha publicado la versión 4 que rompe con

los esquemas anterior y modulariza el lenguaje.

En este proyecto se ha utilizado CSS para definir los estilos de las páginas.

2.10. Log4j

Es una biblioteca código abierto desarrollada en Java por la Apache Software Foundation que permite a los desarrolladores de software escribir mensajes de registro, cuyo propósito es dejar constancia de una determinada transacción en tiempo de ejecución. *Log4j* permite filtrar los mensajes en función de su importancia. La configuración de salida y granularidad de los mensajes es realizada en tiempo de ejecución mediante el uso de archivos de configuración externos.

En este proyecto se ha utilizado para escribir trazas en determinados puntos críticos del código.

3. Arquitectura de la aplicación

En este proyecto se ha desarrollado una aplicación Web Java utilizando la arquitectura Modelo – Vista – Controlador.

El proyecto se han integrado distintas tecnologías para crear la arquitectura.

El backend implementa el servicio que incluye toda la interacción con el sistema *XMPP*.

Esta capa de servicio implementa todos los métodos que utilizarán los controladores para la comunicación con el servidor *XMPP*.

El frontend tiene una estructura particular acorde con la utilización requerida por la arquitectura *JSF 2*.

Los **controladores** son las clases *Managed Beans* propias de *JSF*.

La **vista** está formada por las páginas xhtml propias de *JSF*, así como por los recursos web estáticos: hojas de estilo (con extensión *.css*), ficheros *Javascript* (con extensión *.js*) e imágenes utilizadas en las páginas.

Desde la página web correspondiente al chat se realiza toda la interacción relativa a la funcionalidad del intercambio de mensajes.

Los controladores se han definido con ámbito *Vista* lo que hace que se mantenga la estructura de datos durante todo el tiempo que se permanezca en la misma página.

El **modelo** de datos se actualiza en función de los eventos que realice el usuario mediante la interfaz web pero también se modificará de forma asíncrona cuando reciba mensajes de chat o de cambio de disponibilidad, desde sus contactos, a través de los *listeners*.

Los *listeners* se ejecutan tras la autenticación en el sistema y reciben mensajes propios de *XMPP*.

Para interacción con el servidor *XMPP* la capa de servicio hace uso de la API *Java Smack 4.1.8*. La API consta de diversas librerías Java y, en función de las operaciones y del tipo de aplicación, se requieren determinadas librerías. Por ejemplo, si se requiere utilizar la *vCard*, que es la tarjeta de datos de un contacto, se requiere la librería *smack-extension*. En el caso de esta aplicación, las librerías necesarias para interacción con *XMPP* son las siguientes:



Figura 7. Detalle librerías Smack.

Las librerías necesarias para el uso de *JSF 2.2* utilizando la implementación *Mojarra 2.2* han sido las siguientes:

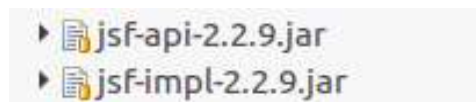


Figura 8. Detalle librerías JSF.

Se han utilizado funcionalidades que no se ofrecen en la distribución *Mojarra 2.2*, por esto, se ha requerido la utilización de *Omnifaces*.

Para el uso de *Omnifaces* se ha requerido incluir la librería *Omnifaces-2.5.1.jar*

Para la generación de trazas para hacer *debug* a distintos niveles de granularidad se ha utilizado la librería *Log4j-1.2.17.jar*.

3.1. Diagrama de clases.

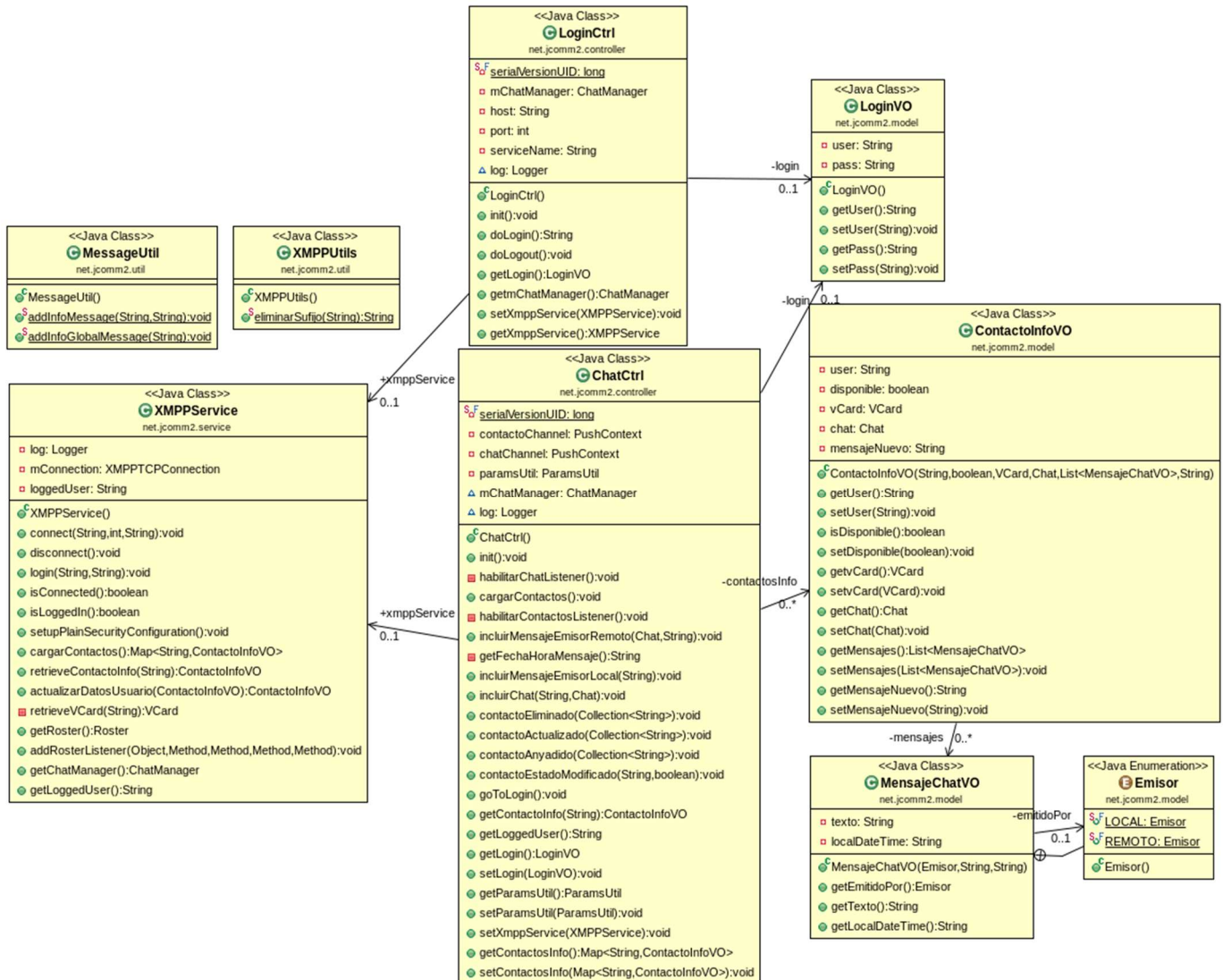


Figura 9. Diagrama de clases.

En el diagrama *UML* se pueden distinguir los dos controladores en la parte central.

Los controladores hacen uso de una instancia de la clase que implementa el servicio *XMPP*.

3.1.1. El controlador *LoginCtrl.class*

Gestiona la autenticación y la desconexión. Para estas dos funciones utiliza la clase del modelo que almacena el usuario y la contraseña. Esta clase se denomina

LoginVO.class.

3.1.2. El controlador *ChatCtrl.class*

Gestiona todas las funciones que permiten mantener la conversación de mensajería instantánea con otro contacto, así como el estado de cada uno de los contactos del usuario.

En el momento de la carga de la clase, se consultan los contactos del usuario, la disponibilidad de estos y la ficha con datos de cada contacto (*vCard*) esta ficha es opcional y contiene un fichero binario con la imagen del *avatar* del usuario, sus números de teléfono, direcciones físicas, dirección de correo electrónico, etc.

En el diagrama UML se puede ver que *ChatCtrl.class* tiene una relación de cero a muchos *ContactoInfoVO.class*. Esta clase tiene los atributos de cada contacto, el estado de su disponibilidad y, a su vez, una relación de cero a muchos *MensajeChatVO.class*, implementada como una lista, que se corresponde con cada uno de los mensajes que se han intercambiado los dos contactos. Esta clase tiene las propiedades requeridas para almacenar el texto del mensaje, la hora y fecha de la creación del mensaje y si lo generó el contacto o el propio usuario.

Cada vez que, por ejemplo, se añade un mensaje a lista, se invoca una instrucción para procesar el panel de mensajes y actualizar la pantalla del navegador de usuario.

3.1.3. Características generales de los controladores

Se ha seguido la recomendación *JSF* de definir un controlador principal para cada vista.

Estos controladores están definidos con ámbito de *vista*, por tanto, mantienen la estructura de datos instanciada durante todo el tiempo en el que el usuario se mantiene en la misma pantalla en la que se invocan. En el caso controlador *chatCtrl*, todos los contactos, el estado de cada uno y la estructura que almacena las conversaciones, las propiedades y los mensajes de estos, se actualiza dinámicamente y, los cambios, se reflejan la estructura representada en el navegador al usuario.

Esta aplicación tiene como característica muy importante la posibilidad de actualización del modelo de datos que gestiona los datos por pantalla a partir de dos tipos de eventos.

- Eventos externos: recibidos a través de *listeners* que se activan a nivel de sesión *HTTP* y que atienden a mensajes que llegan del servidor *XMPP* de forma *asíncrona*.

La aplicación lanza, para cada usuario conectado, dos *listeners* que atenderán a mensajes procedentes del servidor *XMPP*, en el caso de este sistema, de *Openfire*:

- Un listener atiende a cambios en: la disponibilidad de los contactos, la creación, eliminación o actualización de contactos.

Cuando se crea o elimina un contacto, se elimina o se visualiza en la lista de contactos de la columna izquierda.

Cuando cambia su estado de disponibilidad se modifica la imagen una marca de color que aparece junto al contacto.

- Un segundo *listener* gestiona los mensajes de chat de cualquier contacto del usuario. Cuando se recibe un mensaje, si hay una ventana de conversación abierta, se añade el mensaje y, si no hay ventana creada, se genera y se introduce el nuevo mensaje.

Para la actualización de la interfaz del navegador del usuario en tiempo real y de forma asíncrona se ha usado una tecnología de *Push* que permite enviar al navegador un mensaje asíncrono mediante a un *canal* al que se suscribe desde la página *JSF*.

Para implementar la tecnología de *Push* se ha utilizado un componente de la librería de utilidades *JSF* denominada *Omnifaces*.

De esta forma el *listener*, ejecutado en el servidor y asociado a la sesión de un usuario, recibe un mensaje. El mensaje es procesado por el controlador dando lugar a la actualización del modelo de datos. Posteriormente se envía un mensaje por el canal abierto desde la página para que se ejecute un evento que haga una petición al servidor para recoger así la nueva actualización del modelo de datos.

Cuando se carga el controlador *chatCtrl* se inicializa la estructura de datos y se consultan los contactos, su estado y si tienen o no ficha de datos, en las que se incluye, imagen de *avatar*, teléfono, dirección y otros campos propios de cada contacto.

3.1.4. Servicio *XMPPService.class*

Esta clase, como se puede ver en el diagrama de clases, se instancia desde cada uno de los controles para llamar a los distintos métodos que realizan las funciones de interacción con el servidor *XMPP* mediante la librería *Smack*.

Esta clase está instanciada a nivel de sesión *HTTP* de manera que mantiene la instancia de la conexión al servidor *XMPP* durante toda la sesión. Esta conexión es compartida por todos los controladores mediante la inyección de la clase *XMPPService*.

3.2. Casos de uso

3.2.1. Autenticación

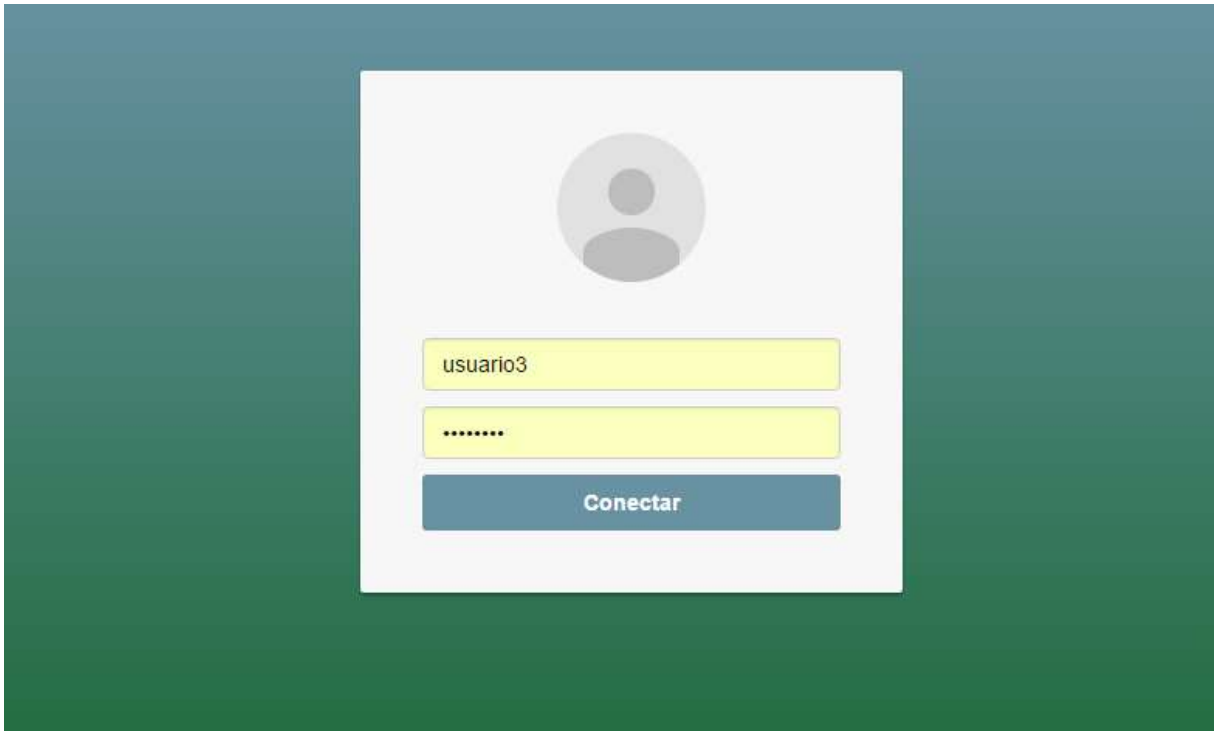


Figura 10. Pantalla de autenticación y desconexión.

La pantalla de entrada inicio de la aplicación es la de autenticación. En la pantalla se presenta un formulario con dos campos de texto para introducir el usuario y contraseña del usuario para autenticar al usuario en el servidor XMPP donde, previamente debe estar registrado con un usuario y contraseña que deberá coincidir con el de entrada.

En caso de autenticación no insatisfactoria se muestra “Usuario o contraseña incorrectos”.

En caso de autenticación satisfactoria se muestra la pantalla de chat.

3.2.2. Acceso a pantalla de chat.



Figura 11. Detalle de la pantalla de chat.

En la pantalla de chat se muestran dos zonas muy diferenciadas. La columna izquierda, con fondo negro, en la que aparecen:

- El usuario autenticado en la sesión. Aparece en color naranja.
- Un listado con todos los contactos del usuario autenticado.

Junto a cada contacto aparece un círculo que, si está conectado se mostrará en verde y, si no está conectado, aparecerá en rojo.

- Un botón con la opción de desconectar la sesión.

3.2.2.1. Abrir conversación con un contacto.

Para abrir una conversación con un contacto, seleccionar el contacto de la lista de contactos. Esta opción solo estaría disponible para contactos conectados. No es posible abrir una ventana de conversación con un contacto no conectado.

Los mensajes que envía el usuario aparecen en el lateral derecho y los mensajes escritos por el contacto, aparecen en el lateral izquierdo de la ventana de conversación.

3.2.2.2. *Escribir mensaje al contacto.*

Para escribir un mensaje a un contacto, escribir el texto en el campo de la zona inferior de la ventana de conversación del contacto. A continuación, seleccionar “Enviar”. En ese momento, el mensaje aparecerá en la ventana de conversación de la página de chat del contacto.

En caso de que el contacto no tuviera abierta la ventana de chat con ese contacto, ésta se abrirá de forma automática.

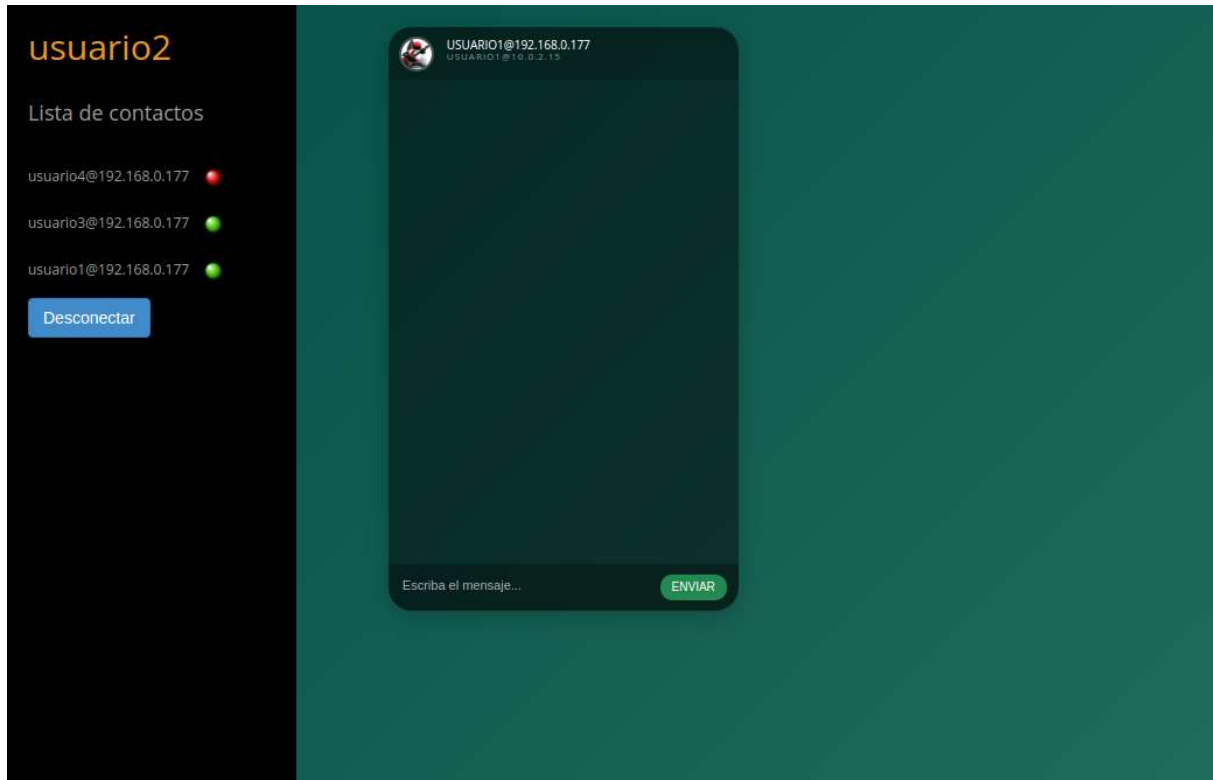


Figura 12. Inicio de conversación con un contacto.

La ventana de chat soporta múltiples ventanas de conversación simultáneas.

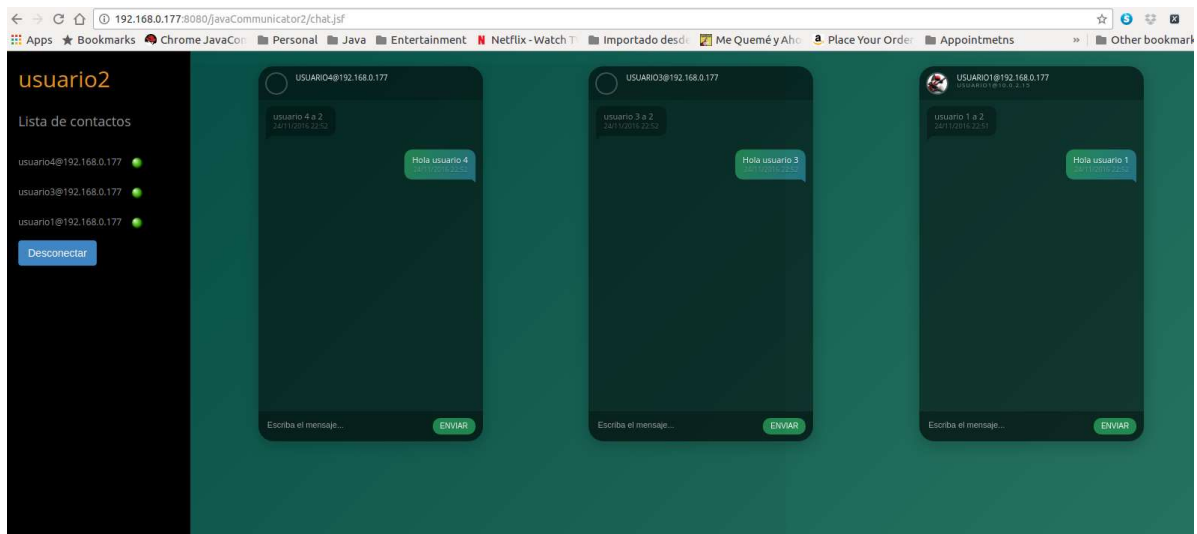


Figura 13. Detalle de múltiples conversaciones.

3.2.2.3. Actualización de listado de contactos y conversaciones en caso de edición, adición o eliminación de contactos.

La lista de contactos se actualiza de forma asíncrona cuando se elimina, se edita o se añade la relación de un contacto mediante otra aplicación o mediante la herramienta de gestión de usuarios del propio servidor XMPP.

3.2.2.4. Desconectar

Seleccionando el botón “*Desconectar*”, se pasa a la página de autenticación, pero con la diferencia de que nombre de usuario no es editable. En la pantalla de autenticación, se mostrarán dos botones: “Ir al chat”, que permite volver a la ventana de chat y “Desconectar” que cierra la sesión del usuario. En este último caso, todos los contactos del usuario reciben un mensaje notificando la desconexión y el estado de la disponibilidad del usuario cambiará en la lista de contactos de la interfaz de cada uno de los contactos.

4. Manual de instalación

La aplicación se distribuye en un fichero *war* (Web Application Archive) archivo de aplicación Web y puede ser ejecutada en cualquier Contenedor de aplicaciones *Java EE* que implemente la especificación *Servlet API 2.5*.

Antes de realizar el despliegue es necesario editar el fichero *war* mediante cualquier aplicación de manejo de fichero comprimidos con tecnología *zip*.

Es necesario configurar dos ficheros:

- *configParams.properties*: en este fichero hay que definir el host, el puerto y el server name del servidor *XMPP* al que se va a conectar. En el caso de este proyecto será *Openfire* y el puerto por defecto es el 5222.
- *log4j.properties*: este fichero contiene la configuración relativa a la librería de trazas de la aplicación. Los parámetros a modificar son:
 - *Log4j.appender.file.File*: para especificar el fichero donde se escribirán las trazas.
 - *Log4j.category.net.jcomm2*: en esta propiedad se ajusta el nivel de trazabilidad que se desea. Los más comunes son: ERROR, INFO, DEBUG.

Se recomienda el servidor de aplicaciones *JBoss 6.1 EAP* para su despliegue.

El despliegue se puede realizar siguiendo el procedimiento propio de cada servidor de aplicaciones.

Una vez desplegada la aplicación, la dirección de acceso será: <http://<dominio>:<8080>/javaCommunicator2>

La página inicial es la de autenticación para el acceso.

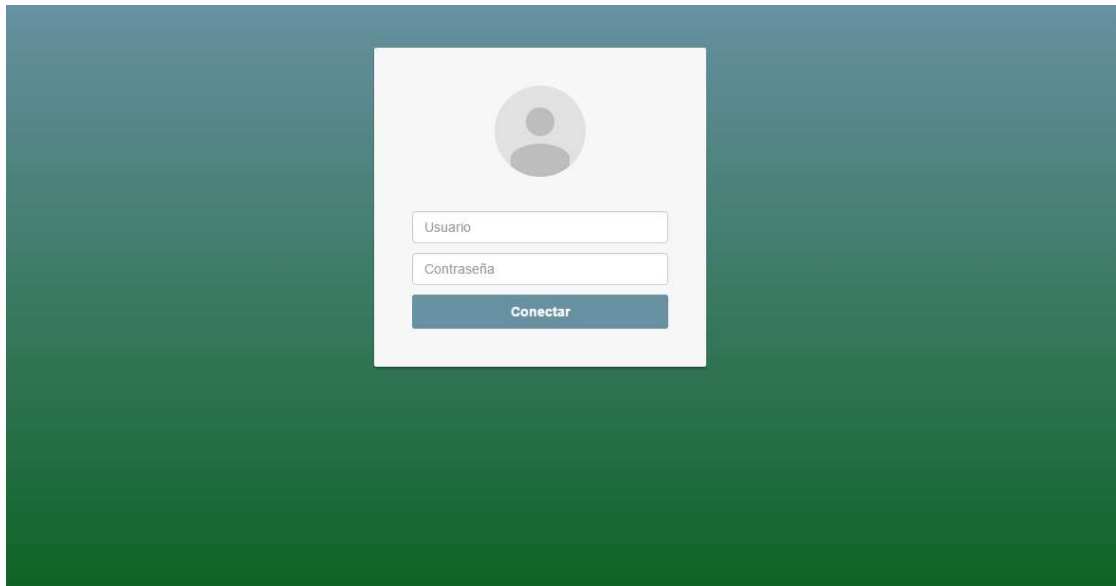


Figura 14. Pantalla de autenticación.



Figura 15. Estructura proyecto en entorno de desarrollo.

4.1. Guía de instalación de *Openfire*

Para este proyecto, *Openfire* se ha instalado en *Ubuntu 16.04*.

La distribución *Openfire 4.0.3* se ha descargado de la página oficial en formato *.deb* . Haciendo doble *click* sobre el fichero comenzará el proceso de instalación.

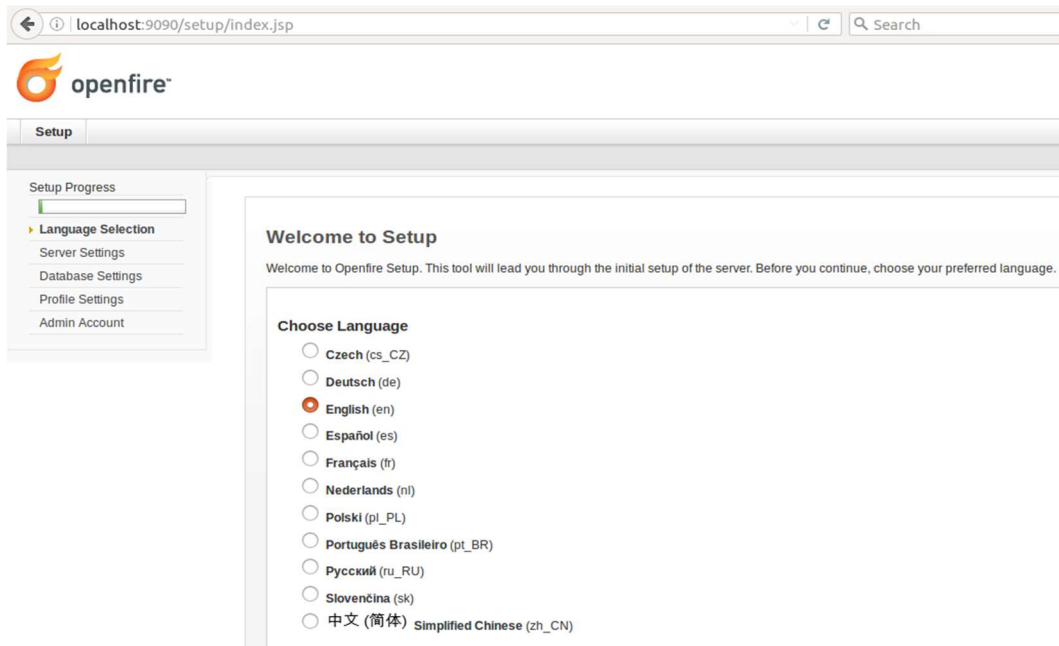


Figura 16. Asistente configuración *Openfire* (selección de idioma).

Una vez terminada la instalación, la aplicación de administración es accesible desde el navegador web en la URL <http://<dominio>:9090> se presentará un asistente para guiar la configuración.

1. Seleccionar idioma de la aplicación.
2. Configuración de las propiedades del servidor.

El *domain* es el nombre con el que identificarán los contactos para cada usuario. No tiene por qué ser la dirección IP del servidor. Puede ser el nombre de un dominio.

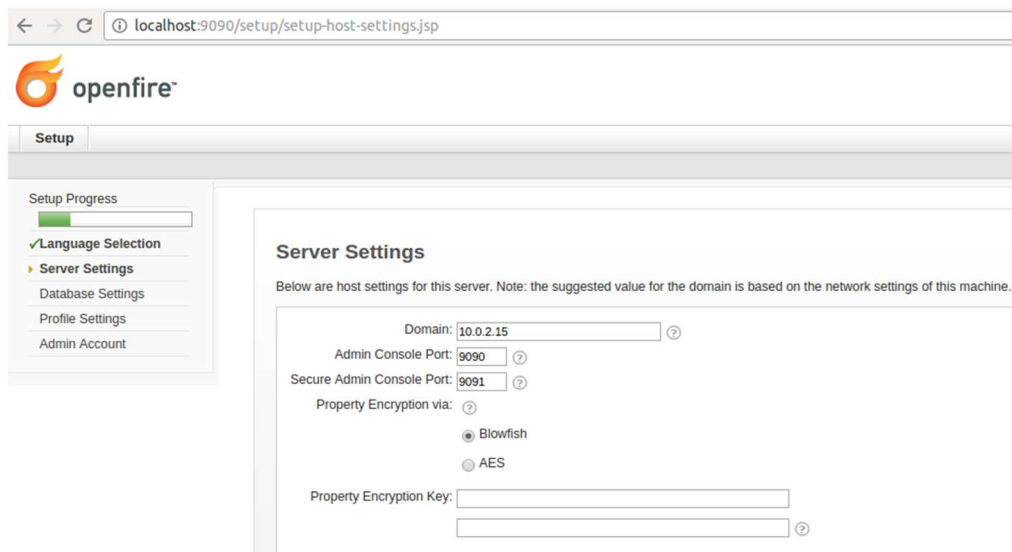


Figura 17. Asistente configuración Openfire (opciones de servidor).

3. Configuración de la base de datos donde se creará el esquema de datos de Openfire.

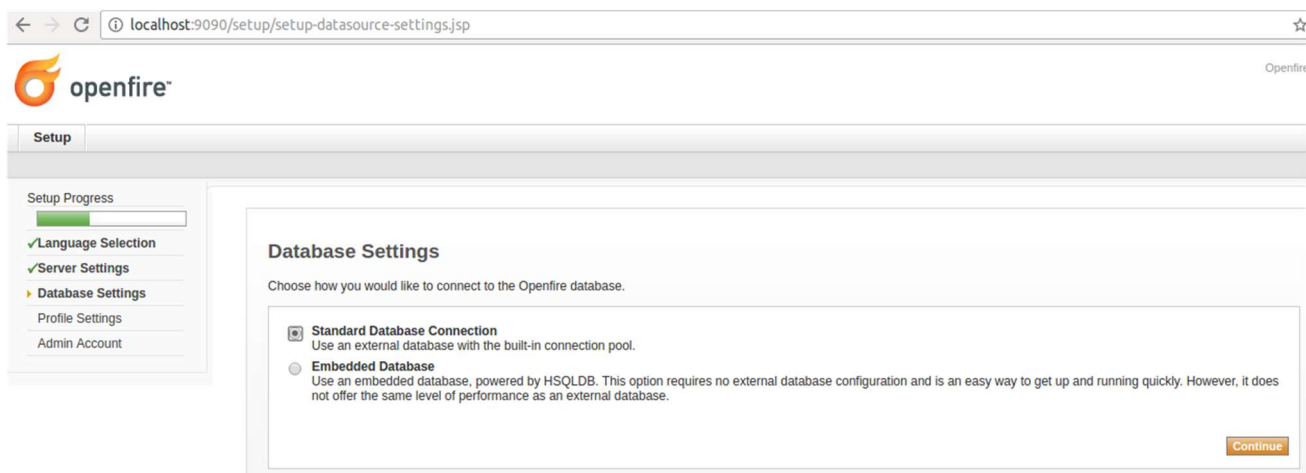


Figura 18. Asistente configuración Openfire (opciones de Base de Datos).

- a. Se puede seleccionar utilizar una Sistema Gestor de Bases de Datos embebido o utilizar una conexión estándar a base de datos. En este caso he utilizado el SGBD MySQL.
- b. Para instalar *MySQL* Server se ejecuta en el terminal: `sudo apt-get install mysql-server` y en el script de instalación solicitará establecer la contraseña de root.
- c. Crear el esquema 'openfire' y el usuario 'openfire' con permisos *DML* y *DDL* sobre el esquema.
- d. Configuración de las credenciales en *Openfire*. En la captura se han introducido las credenciales para conectar como root pero es

aconsejable user un usuario específico que tenga acceso únicamente al esquema “*openfire*”

The screenshot shows the Openfire setup wizard at the 'Database Settings - Standard Connection' step. The browser address bar shows 'localhost:9090/setup/setup-datasource-standard.jsp'. The Openfire logo is in the top left, and 'Openfire' is in the top right. A 'Setup' tab is active. On the left, a 'Setup Progress' bar shows the current step is 'Database Settings'. Below it, a list of steps includes 'Language Selection', 'Server Settings', 'Database Settings', 'Profile Settings', and 'Admin Account'. The main content area is titled 'Database Settings - Standard Connection' and contains the following text: 'Specify a JDBC driver and connection properties to connect to your database. If you need more information about this process please see the database documentation distributed with Openfire.' Below this is a note: 'Note: Database scripts for most popular databases are included in the server distribution at [Openfire_HOME]/resources/database.' The form fields are: 'Database Driver Presets' (dropdown menu set to 'MySQL'), 'JDBC Driver Class' (text input 'com.mysql.jdbc.Driver'), 'Database URL' (text input 'jdbc:mysql://10.0.2.15:3306/openfire?rewriteBatchedStatements'), 'Username' (text input 'root'), 'Password' (password input field with masked characters), 'Minimum Connections' (text input '5'), 'Maximum Connections' (text input '25'), and 'Connection Timeout' (text input '1.0' followed by 'Days'). A note at the bottom right states: 'Note, it might take between 30-60 seconds to connect to your database.'

Figura 19. Asistente de configuración Openfire (credenciales de Base de Datos).

The screenshot shows the Openfire setup wizard at the 'Profile Settings' step. The browser address bar shows 'localhost:9090/setup/setup-datasource-standard.jsp'. The Openfire logo is in the top left, and 'Openfire' is in the top right. A 'Setup' tab is active. On the left, a 'Setup Progress' bar shows the current step is 'Profile Settings'. Below it, a list of steps includes 'Language Selection', 'Server Settings', 'Database Settings', 'Profile Settings', and 'Admin Account'. The main content area is titled 'Profile Settings' and contains the following text: 'Choose the user and group system to use with the server.' Below this are three radio button options: 'Default' (selected), 'Only Hashed Passwords', and 'Directory Server (LDAP)'. The 'Default' option description is: 'Store users and groups in the server database. This is the best option for simple deployments.' The 'Only Hashed Passwords' option description is: 'Store only non-reversible hashes of passwords in the database. This only supports PLAIN and SCRAM-SHA-1 capable clients.' The 'Directory Server (LDAP)' option description is: 'Integrate with a directory server such as Active Directory or OpenLDAP using the LDAP protocol. Users and groups are stored in the directory and treated as read-only.' A 'Continue' button is located at the bottom right of the form.

Figura 20. Asistente de configuración Openfire (Opciones de perfil).

Carga de usuario y grupos en *Openfire*. En esta opción permite importar desde LDAP. Almacenar en Base de Datos solo la contraseña en hashes o almacenar los usuario y grupos en la base de datos. La última opción es la recomendada.

4. Establecer usuario y contraseña del usuario inicial que tendrá acceso a la herramienta de administración. En este caso se ha configurado admin/admin.

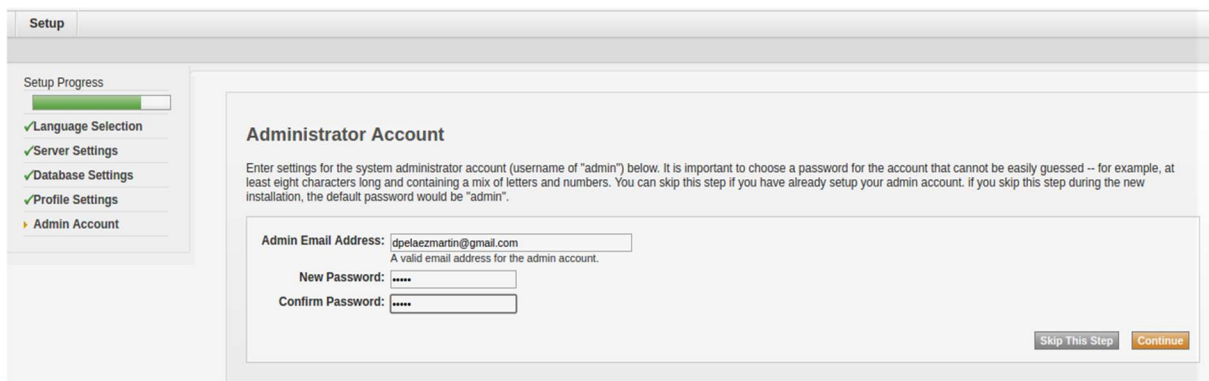


Figura 21. Asistente de configuración Openfire (cuenta de administrador).

5. Detalle de los puertos que habilita Openfire y la función de cada uno.

Interface	Port	Type	Description
All addresses	5222	Client to Server	The standard port for clients to connect to the server. On this port plain-text connections are established, which, depending on configurable security settings , can (or must) be upgraded to encrypted connections.
All addresses	5223	Client to Server	The port used for clients to connect to the server using the old SSL/TLS method. Connections established on this port are established using a pre-encrypted connection. This type of connectivity is commonly referred to as the "old-style" or "legacy" method of establishing encrypted connections. Configuration details can be modified in the security settings .
All addresses	7070	HTTP Binding	The port used for unsecured HTTP client connections.
All addresses	7443	HTTP Binding	The port used for secured HTTP client connections.
All addresses	5269	Server to Server	The port used for remote servers to connect to this server. Connections established on this port are established using a pre-encrypted connection. This type of connectivity is commonly referred to as the "old-style" or "legacy" method of establishing encrypted connections. Configuration details can be modified in the security settings .
All addresses	5275	External Components	The port used for external components to connect to the server. On this port plain-text connections are established, which, depending on configurable security settings , can (or must) be upgraded to encrypted connections.
All addresses	5276	External Components	The port used for external components to the server using the old SSL/TLS method. Connections established on this port are established using a pre-encrypted connection. This type of connectivity is commonly referred to as the "old-style" or "legacy" method of establishing encrypted connections. Configuration details can be modified in the security settings .
All addresses	5262	Connection Manager	The port used for connection managers to connect to the server. On this port plain-text connections are established, which, depending on configurable security settings , can (or must) be upgraded to encrypted connections.
All addresses	5263	Connection Manager	The port used for connection managers to the server using the old SSL/TLS method. Connections established on this port are established using a pre-encrypted connection. This type of connectivity is commonly referred to as the "old-style" or "legacy" method of establishing encrypted connections. Configuration details can be modified in the security settings .
All addresses	9090	Admin Console	The port used for unsecured Admin Console access.
All addresses	9091	Admin Console	The port used for secured Admin Console access.
All addresses	7777	File Transfer Proxy	The port used for the proxy service that allows file transfers to occur between two entities on the XMPP network.
All addresses	5229	Flash Cross Domain	Service that allows Flash clients connect to other hostnames and ports.

[Edit Properties](#)

Figura 22. Detalle de puertos y configuración.

6. Detalle de la configuración del final del servidor. Es importante destacar que el "Server Name" se debe especificar para identificar al contacto de un usuario cuando se realiza la inserción de los contactos.

Server Manager | Server Settings | TLS/SSL Certificates | Media Services

Server Information

- System Properties
- Language and Time
- Clustering
- Cache Summary
- Database
- Logs
- Email Settings
- Security Audit Viewer

Server Information

Below you will find server information, ports being used and latest news about Openfire.

Server Properties

- Server Uptime: 54 minutes -- started Oct 28, 2016 11:08:31 PM
- Version: Openfire 4.0.3
- Server Directory: /usr/share/openfire
- Server Name: 10.0.2.15

Environment


- Java Version: 1.8.0_91 Oracle Corporation -- OpenJDK 64-Bit Server VM
- Appserver: jetty/9.2.z-SNAPSHOT
- Host Name: david-VirtualBox
- OS / Hardware: Linux / amd64
- Locale / Timezone: en / Central European Time (1 GMT)
- Java Memory  43.21 MB of 1235.44 MB (3.5%) used

Figura 23. Openfire, descripción de detalles del servidor.

7. Alta de usuarios.

Para dar de alta usuarios acceder a la pestaña 'Users/Groups'.

Server | **Users/Groups** | Sessions | Group Chat | Plugins

Users | Groups

User Summary
Create New User
User Search

User Summary

Total Users: 5 -- Sorted by Username -- Users per page: 100
















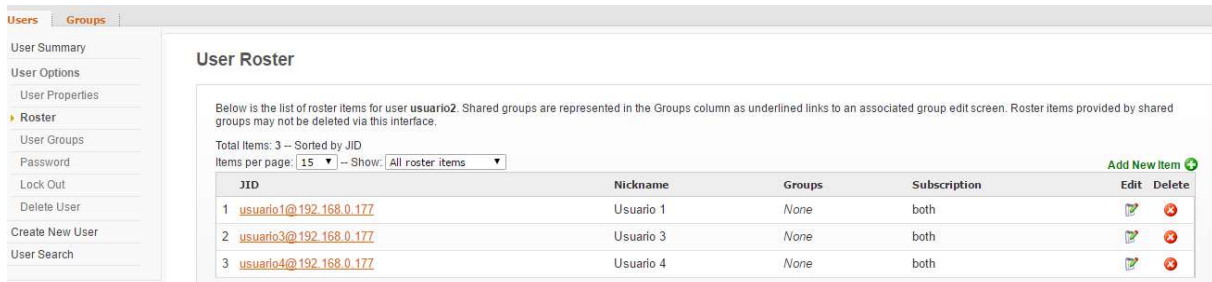
Online	Username	Name	Groups	Created	Last Logout	Edit	Delete
1	 admin	Administrator	None	Oct 28, 2016			
2	 usuario1	Usuario 1	None	Nov 20, 2016			
3	 usuario2	Usuario 2	None	Nov 20, 2016			
4	 usuario3	Usuario 3	None	Nov 20, 2016	51 minutes		
5	 usuario4	Usuario 4	None	Nov 22, 2016			

Figura 24. Openfire, listado de usuarios.

- Para definir los contactos de un usuario, seleccionar el usuario y en el menú que aparece seleccionar la sección 'Roster'. Para añadir a un contacto se debe registrar con su nombre de usuario seguido de '@' y el nombre del 'Server Name' de *Openfire*.



The screenshot shows the 'User Roster' page in the Openfire web interface. The sidebar on the left contains navigation links: User Summary, User Options, User Properties, Roster (highlighted), User Groups, Password, Lock Out, Delete User, Create New User, and User Search. The main content area is titled 'User Roster' and contains a table of roster items for user 'usuario2'. The table has columns for JID, Nickname, Groups, Subscription, Edit, and Delete. There are three items listed, all with a 'both' subscription and 'None' groups.

JID	Nickname	Groups	Subscription	Edit	Delete
1 usuario1@192.168.0.177	Usuario 1	None	both		
2 usuario3@192.168.0.177	Usuario 3	None	both		
3 usuario4@192.168.0.177	Usuario 4	None	both		

Figura 25. Openfire, listado de contactos.

- Una vez insertado editar el contacto y modificar el modo de suscripción a 'Both'. Esto hace que el servidor envíe mensajes al contacto con información del usuario y viceversa. La relación como 'contacto' debe estar definida en los Rosters de ambos usuarios. Debe ser bidireccional.

4.2. Instalación de cliente de escritorio basado en XMPP

Un ejemplo de cliente de escritorio de mensajería instantánea basada en XMPP es *Empathy*.

Para instalarlo en Ubuntu: `sudo apt-get install empathy`

5. Conclusiones

Se ha desarrollado el proyecto cumpliendo los objetivos establecidos.

Para crear el proyecto se ha realizado un estudio del arte de la tecnología *XMPP* en general y más en concreto del software libre que había disponible para realizar el proyecto.

Se ha utilizado *Openfire* como servidor *XMPP* por ser un servidor flexible y de libre distribución. Se ha estudiado su funcionamiento, su estructura de datos, el proceso de instalación y configuración y parte de la administración: creando usuarios y relaciones entre ellos para desarrollar las pruebas de la aplicación web.

Se podría haber utilizado el servidor público para test <https://xmpp.net> pero se ha optado por *Openfire* para adquirir más conocimientos y por ofrecer la posibilidad de instalarse en Intranets. También ofrece la posibilidad de sincronización con *LDAP* lo que permite ampliar este proyecto en el futuro.

En este proyecto se planteaba como principal reto la integración de las distintas tecnologías necesarias para que una aplicación Web pueda servir interfaces de mensajería instantánea, que requieren actualización asíncrona, en un navegador web y hacer todo dependiente de la sesión *HTTP*.

Cabe mencionar que ha resultado muy difícil encontrar ejemplos de uso de la API *Smack 4* porque se ha realizado un cambio de arquitectura en el paso de versión de la 3 a la 4 , aún no tiene un uso masivo debido a su reciente publicación.

Los ejemplos existentes en la Web hacen referencia, en su mayoría a la versión 3. El proyecto se comenzó hace 2 años y la librería en se tenía basado el desarrollo era la versión 3.

Se podría haber desarrollado todo el proyecto utilizando la versión 3 y una versión de *Openfire* inferior, para evitar incompatibilidades sobre la especificación de *XMPP* que implementan. Finalmente se ha optado por utilizar las últimas versiones de todos los componentes para que el sistema pueda ser ampliado en un futuro y también para que tarde más tiempo en quedar obsoleto.

La integración de una tecnología como *JSF*, con un ciclo de vida muy complejo, de manera que los controladores instancien *listeners* a nivel de sesión y que a su vez, estos controladores mantengan una estructura de datos que permita disponer la información en pantalla y el hecho de utilizar una tecnología que, de forma asíncrona, actualice la interfaz del navegador en función de eventos asíncronos, ha sido muy satisfactorio aunque en muchas ocasiones se ha cuestionado su viabilidad.

Por otro lado, me ha resultado especialmente complejo el diseño web a partir de plantillas CSS modificadas y adaptadas a la funcionalidad implementada con *JSF*. El campo de las *CSS* y *Javascript* está evolucionando muy rápido y la curva de aprendizaje es bastante importante.

En cuanto a *CSS*, en particular, la publicación de la especificación *CSS4* y las técnicas como *LESS* y de los ficheros *SCSS* que permiten el anidamiento de las especificaciones de estilos, dan mucha complejidad y potencia a las hojas de estilo.

En cuanto a *Javascript*, está evolucionando muy rápido en los últimos años, gracias a los motores de ejecución como *Node.js*, que le permiten sacarlo del entorno de ejecución de los navegadores y de *frameworks* como *AngularJS*, *JQuery*, etc. El trabajo como *frontend developer* en exclusiva está cada vez más justificado debido al grado de especialización que se requiere.

La tendencia actualmente es el desarrollo de aplicaciones con el uso de microservicios en la parte del servidor y el resto de la ejecución se realiza en *Javascript* y tiene lugar en el navegador del usuario.

El proyecto me ha permitido adquirir conocimientos sobre la tecnología *JSF 2.2*. El correcto uso de los *Managed Beans* es esencial para crear una arquitectura que, en el futuro reduzca el coste de mantenimiento. *JSF* ofrece soluciones para la gran cantidad de situaciones que surgen en la interacción con la aplicación desde un navegador Web.

JSF hace uso de *AJAX* para mejorar la experiencia del usuario, pero es necesario comprender bien las fases del ciclo de vida de *JSF* para no tener problemas en la gestión de los eventos en el navegador.

La librería *Omnifaces* ha facilitado la implementación por ofrecer utilidades que, por su evidente necesidad, muy probablemente, estarán incluidas en la próxima especificación de *JSF*. Entre las muchas utilidades que implementa, el componente para gestionar la tecnología *Push*, es muy bueno. Ofrece gran cantidad de modificadores para recoger la mayoría de las situaciones que se pueden presentar.

Otra utilidad de *Omnifaces* de la que se ha hecho uso en la aplicación es la de la generación de una imagen gráfica en el cliente a partir de un flujo de bytes. Generalmente se requiere una *request* exclusiva para descargar el binario tras haber cargado la página. El componente *GraphicImage* de *Omnifaces* permite implementar esa funcionalidad de forma sencilla. La documentación de *Omnifaces* es muy explicativa y facilita mucho conocer el funcionamiento de los componentes que ofrece.

6. Líneas futuras

La estructura de datos que se ha utilizado para gestionar los datos es muy robusta y escalable. Podrían incluirse muchas más funcionalidades a la aplicación con cambios mínimos en la estructura de datos. Algunas funcionalidades que se podrían añadir son:

- Registro de nuevos usuarios: alta de nuevo usuario en dos fases con envío de email para verificar autenticidad.
- Creación de relaciones entre usuarios: Implementar la funcionalidad permita a un usuario registrado buscar otros usuarios y enviarle una petición 'petición de amistad' que, si es confirmada, permita incluirlo como contacto.
- Gestionar el bloqueo temporal de usuarios.
- Establecer el estado de disponibilidad de forma personalizada.
- Cerrar ventanas de chat con un botón.
- Marcar en la columna de contactos el número de mensajes por contacto, que el usuario tiene pendiente de leer.
- Incluir la posibilidad de envío de pequeñas imágenes.
- Definir en la pantalla de autenticación los datos del servidor al que se desea conectar, para que, la misma interfaz permita conectarse a distintos dominios.
- Implementar soporte para conversaciones en grupo.
- Permitir ver la ficha de un usuario en una ventana.
- Permitir crear una ficha del usuario y almacenarla en el servidor. Guardando datos como: la imagen del avatar, teléfono, email, dirección postal, etc.
- Almacenar histórico de conversaciones para mostrarlas cuando se abre la ventana de chat de un contacto. De forma que permita continuarla.
- Entrega de mensajes en diferido. Cuando el contacto se conecta, recibirá todos los mensajes que sus contactos le hayan enviado mientras él estaba desconectado.
- En lugar de añadir funcionalidad, enfocar la aplicación a la gestión de elementos de Internet de las Cosas (IoT) de manera que los dispositivos envíen mensajes con el estado y el usuario, en lugar de tener contactos, sean dispositivos con sensores comunicando información.

7. Bibliografía

- Mastering Java Server Faces 2.2
- XMPP: [HTTPs://XMPP.org/](http://XMPP.org/)
- *Java EE 7*
- *Java EE 7 Development with WildFly*
- *Java EE 7 Developer Handbook*
- Tutorial oficial de *Java EE 7*.
- Especificación JSF 2. <http://download.oracle.com/otndocs/jcp/jsf-2.0-fr-full-oth-JSpec/>
- Logj4 <https://logging.apache.org/log4j/1.2/manual.html>
- Omnifaces Push <http://showcase.omnifaces.org/push/socket>
- Descarga de imagen a partir de binario:
<http://showcase.omnifaces.org/components/graphicImage>
- Instalación Openfire:
<http://download.igniterealtime.org/openfire/docs/latest/documentation/install-guide.html>
- Librería Smack:
<https://www.igniterealtime.org/projects/smack/documentation.jsp>
- <http://stackoverflow.com/>