- Proyecto Final de Carrera - Implantación de una comunicación chat con django

Autor: Miguel Ángel Cuevas Cepeda
Tutor: Don Juan Antonio Ternero Muñiz

11 de julio de 2012

Departamento de Ingeniería Telemática
Ingeniería de Telecomunicación
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

AGRADECIMIENTOS

Quiero agradecer a Don Juan Antonio Ternero Muñiz, profesor e investigador en el Departamento de Ingeniería de Telemática de la Escuela Superior de Ingeniería de Sevilla, la realización del proyecto. A mi familia el apoyo que me han dado y en especial a mi novia Sara, por confiar en mí y darme ánimos en todo momento.

Índice general

1 Conocimientos previos

1.1. Introducción

Este documento es un proyecto final de carrera realizado por *Miguel Ángel Cuevas Cepeda* con la colaboración del tutor, Don *Juan Antonio Ternero Muñiz*. La elaboración del proyecto tiene por finalidad dos objetivos:

1. La primera es la elaboración de un documento que permita a *Miguel Ángel Cuevas Cepeda* realizar el Proyecto Final de Carrera y terminar sus estudios de Ingeniería de Telecomunicación.

2. Estudio y desarrollo de un cliente y un servidor de mensajería instantánea mediante una interfaz web utilizando la tecnología *django*[1].

A continuación se comenta la estructura de la memoria:

Capítulo_1 Es una introducción al *Proyecto Final de Carrera*, se recogen los objetivos y funcionalidades de la aplicación implementada.

Capítulo_2 Instalación, configuración e investigación de los programas empleados.

Capítulo_3 Estudio con *Wireshark* para el análisis de las comunicaciones.

Capítulo_4 Implementación de la aplicación *chat* para *django*.

Capítulo_5 Programas alternativos analizados e instalados pero no utilizados para el desarrollo del proyecto.

Hoy en día hay muchas alternativas de chat, también llamado clientes de mensajería instantánea, por ejemplo, *windows live messenger 1.1a,* clientes de mensajería libres (open source) como *pidgin* 1.1b, *eMesene* 1.1c, etc.

[1] Mas adelante veremos que es y que servicios proporciona *django.*

(a) Windows live Messenger (b) Pidgin (c) eMesene

Figura 1.1: Clientes de mensajería

Estos clientes tienen como mínimo un requisito esencial, su instalación previa en un sistema operativo, en cambio, hay otros clientes de mensajería que no necesitan una instalación. Por ejemplo podemos mencionar, los servicios de correos electrónicos como *hotmail* y *gmail*, que utilizan una interfaz web para la transmisión y recepción de mensajes. Si se inicia una sesión, estos servicios de correos electrónicos pone a disposición del usuario herramientas para tal fin. Se recalca de nuevo, que no ha habido ninguna instalación de una aplicación, es el navegador web el que proporciona las herramientas necesarias para una comunicación del tipo que se está comentando. Este nueva forma de utilizar el navegador web como cliente de mensajería es una tecnología en expansión.

Este proyecto tiene por propósito final la implantación de una comunicación de un cliente de mensajería mediante una interfaz web utilizando como servidor *django*, que es un *framework web* de código abierto escrito en *Python*.

1.2. Funcionalidad del chat

1.2.1. Compatibilidad

El cliente[2] va a utilizar, como interfaz de comunicación, un navegador web. Podrá funcionar sobre cualquier plataforma y sistemas operativos *Linux*, *Microsoft Windows* y *Macintosh*, en cualquiera de sus versiones. En principio no hay ninguna restricción en el navegador web que se vaya a utilizar para ejecutar la aplicación cliente (se quiere que su uso no se vea restringida por éste).

En el proyecto se han utilizado los siguientes navegadores web:

[2] A partir de ahora cuando se hable de *cliente*, se hará alusión a un *cliente de mensajería instantáneo o chat*.

- Mozilla Firefox for Ubuntu (v.3.5.9)
- Google Chrome (v.9.0.597.98)
- Internet Explorer (v.7 & v.8)

1.2.2. Conectividad

Conexión del cliente al servidor (*django*), se puede iniciar desde cualquier punto de la red o internet, basta con tener acceso a una conexión TCP/IP (con un puerto dado[3]).

1.2.3. Seguridad de la conectividad

En principio sobre el tema de seguridad no se ha entrado en profundidad. Una posible mejora en este proyecto podría darse en este ámbito, posibilitando una comunicación más segura, utilizando los mecanismos de cifrado.

1.2.4. Funcionalidades básicas que debe de tener un cliente de mensajería

Las funcionalidades básicas utilizadas, las avanzadas se ven en el siguiente punto, son las que se detallan a continuación. Las que no se han desarrollado se especifican en la tabla 1.1, en ella se resume las funcionalidades analizadas.

- Texto síncrono para una comunicación directa e instantánea en línea con cualquier usuario conectado al sistema. Permite el envío y recepción de mensajes escritos, con características gráficas, como:
  - tipografía de color (no contemplado en este proyecto, posibilidad de ampliación)
  - distinto tamaño (no contemplado en este proyecto, posibilidad de ampliación)
- Espacio reservado donde aparezca el texto que se está escribiendo, así como un botón para aceptar el envío de dicho texto.
- Botones para iniciar otros módulos: una videollamada, una transferencia de datos y mensajes privados entre usuarios.

[3]Más adelante se describirá con mayores detalles las conexiones y protocolos que se utilizarán.

- Una pestaña “Setup” para entrar como “Administrador” y añadir o eliminar usuarios del chat.

- Información adicional en el chat como la fecha y hora de la última recepción de un mensaje escrito otro usuario.

Todas las características anteriores se han contemplado para este proyecto. Se irá describiendo con mayor profundidad en los siguientes capítulos.

1.2.5. Funcionalidades avanzadas

- Incluir una foto personal del usuario en la ventana de chat.

- Soporte de direcciones URL, emoticonos, etc.

- Todas las sesiones pueden quedar registradas en la base de datos para ser recuperadas posteriormente y ponerlas a disposición del que las requisite (problemas y disputas entre usuarios).

- Alertas de eventos. Una vez que se ha registrado un usuario en el chat, el servidor puede notificarle de tareas pendientes que anteriormente se hayan guardado en la base de datos o de avisos por parte de otros usuarios. Este punto es muy interesante, no contemplado en el proyecto con posibilidad de ampliación. Aunque el evento de ingreso de un usuario al chat sí que está contemplado.

- Interfaz del chat distribuida en canales: el administrador es el creador de los canales, cuando un usuario entra en la aplicación, puede elegir a que canal conectarse.

- Un usuario puede leer los comentarios de los demás si éste pertenece al mismo canal que estos. Un usuario que no pertenezca al canal, no podrá leerlo. La forma de permitir una comunicación entre dos usuarios es que estén conectados al mismo canal, o si no lo están, se ha desarrollado un mecanismo de comunicación indirecto unidireccional mediante mensajes privados entre usuarios de distinto canal.

- Por último la posibilidad de guardar la conversación en un archivo para la posterior revisión del mismo, ver nota 1.

1.2.6. Resumen de las funcionalidades

Funcionalidades / Características de un chat vistas hasta ahora:

X	Uso de cualquier sistema operativo
X	Uso de una interfaz web para la comunicación
X	Control de perfiles
X	Emoticonos
X	Rápido aprendizaje y gran facilidad de uso
X	Interactividad "instantánea"
X	Comunicación dividida en canales públicos o privados
-	Conectividad segura
-	Agenda personal, notificación de eventos
X	Posibilidad de guardar conversaciones
-	Estandarización en las comunicaciones
X	Interacción con otros módulos (videollamada, mensajes privados)

Cuadro 1.1: Resumen de las características más importantes

En el cuadro 1.1 se ha marcado con *X* las funcionalidades ofrecidas en este proyecto, y con - las posibles mejoras.

1.3. Fundamentos previos

Esta sección va dedicada a los nuevos conocimientos adquiridos a lo largo del proyecto. Es muy importante resaltar que gran parte del tiempo que se le ha dedicado a este proyecto ha sido a la obtención de nuevos lenguajes de programación y a la investigación.

En el caso de los lenguajes de programación se completó su lectura con los ejercicios que iban planteándose. En el CD Suplementario hay una carpeta por cada lenguaje con las soluciones de cada ejercicio.

Los nuevos conocimientos adquiridos durante el desarrollo del proyecto son los siguientes:

- Lenguaje de programación *JavaScript.*
  - Ver referencia [9]

  - Enlace web al libro utilizado*:*
    http://www.librosweb.es/javascript/index.html

JavaScript se utiliza en el lado cliente, en este caso, implementado en el navegador web permitiendo mejoras en la interfaz de usuario y posibilitando una web dinámica.

- Lenguaje de programación *Python*.

  - Ver referencia [10]
  - Enlace web al libro utilizado:
    http://mundogeek.net/tutorial-python/

Se recuerda que el proyecto tiene por objetivo el uso de la tecnología *django*. *django* está escrito en código *python*, es por ello el estudio del mismo. *Python* es un lenguaje de programación de alto nivel cuya filosofía hace hincapié en una sintaxis muy limpia y que favorezca un código legible.

- Lenguaje de programación *XHTML* para la creación y modificación de páginas web.

  - Ver referencia [11]
  - Enlace web al libro utilizado:
    http://www.librosweb.es/xhtml/index.html

- Uso de la tecnología *AJAX*.

  - Ver referencia [12]
  - Enlace web al libro utilizado:
    http://www.librosweb.es/ajax/index.html

- *Framework Web django*, provee de una infraestructura de programación para aplicaciones.

  - Ver referencia [13]
  - Enlace web al libro utilizado:
    http://es.scribd.com/doc/61814981/El-Libro-de-Django

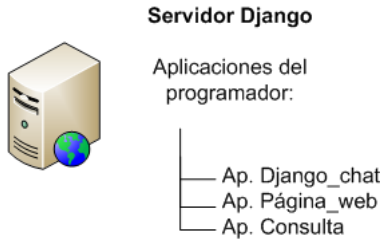

Figura 1.2: Aplicaciones del programador en un servidor django

Para ésta última referencia se adelanta una característica, su funcionamiento, para que que el lector tenga una noción de su puesta en marcha, aunque se profundizará más en los siguientes capítulos. El servidor *django* interactúa con aplicaciones, ver figura 1.2, que el programador ha escrito, dejando a *django* los trabajos comunes en el desarrollo de aplicaciones web. La aplicación web utilizada se llama *django_chat*, por lo tanto la comunicación queda como sigue, cliente (en una interfaz web) y el servidor chat *django* que interactúa con la aplicación *django_chat,* ver figura *1.3.*

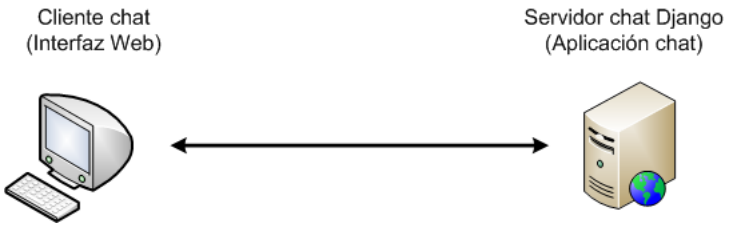

Figura 1.3: Comunicación cliente <-> servidor

2 Instalación y configuración

Éste capítulo trata la instalación y la configuración del cliente. Además, se explica el proceso de instalación del servidor así como las dependencias del mismo.

La instalación del servidor se ha realizado en una única máquina. Si se quiere instalar en otra, los pasos serán los mismos, aunque hay que tener cuidado con la arquitectura del ordenador utilizado, ya que, éste puede ser distinta. Se remite al lector ver su arquitectura y actuar en consecuencia. Las referencias a los binarios (ejecutables), que aquí se comentan, para la instalación se refieren a los que se han utilizado para realizar este proyecto.

Con respecto al cliente, el único requisito es tener instalado un navegador web[1].

Las especificaciones de la arquitectura se nombra a continuación[2]:

Nombre de la especificación	Tipo
Sistema Operativo	Ubuntu 9.10 - *Karmic Koala* -
Tipo de Arquitectura	i386

Cuadro 2.1: Especificaciones de la arquitectura utilizada en este proyecto

2.1. Requisitos software

Estas son las aplicaciones que se han instalado para el funcionamiento del servidor:

- Instalación de Python (v.2.6.4): Subsección 2.2.1
- Instalación de Django (v.1.1.1): Subsección 2.2.2
- Instalación de Apache (v2.2.12): Subsección 2.2.3

[1]Para este proyecto es indiferente el navegador web utilizado, aunque más adelante se hablará que con el navegador web *internet explorer*, en cualquiera de sus versiones, se tiene la posibilidad añadida de guardar las conversaciones.

[2]La distribución de Ubuntu 9.10 fue publicada en Octubre de 2009 y soportada hasta Abril de 2011. Este proyecto tuvo una duración de 6 meses correspondiente al período comprendido entre febrero - julio 2011, por ello la distribución linux utilizada.

2.2. Investigación, instalación y configuración de django

2.2.1. Instalación de Python

El código de *django* está 100 % escrito en puro *python* así que lo primero que se necesita es instalar *python*. *Python* se encuentra en los paquetes de cualquier sistema operativo basado en GNU/Linux. Para la instalación de *python* en ubuntu (debian) 9.10 se teclea lo siguiente en un terminal[3]:

```
# sudo apt-get install python
```

Se utilizará la versión más estable, versión 2.6.4. Para comprobar que la instalación de *python* se realizó con éxito, basta entrar en la consola de *python*:

```
# python
# Python 2.6.4 (r264:75706, Dec  7 2009, 18:45:15)
[GCC 4.4.1] on linux2 Type "help", "copyright", "credits" or
"license" for more information.
>>>
```

2.2.2. Instalación *django*

La página web oficial de *django (web framework)* se encuentra en:

https://www.djangoproject.com/

Se puede encontrar visitando este enlace documentación, guias, los binarios para la instalación en un sistema operativo, así como, un foro para la resolución de dudas, comentarios, fallos, etc. La página oficial está en ingles, existe también la posibilidad de encontrar la web *django* en español:

http://django.es/

Al igual que antes, en ésta página web se encuentra toda la documentación en español, los binarios para la instalación, discusión de dudas, foros y charlas sobre este tema.

[3] Cuando se trate temas relacionados con la intalación, configuración o edición de ficheros se necesitan privilegios de administrador.

Para la instalación de *django* en *Ubuntu 9.10* solo hay que teclear lo siguiente en un terminal:

```
# sudo apt-get install python-django
```

2.2.3. Instalación de *Apache* y módulos de comunicación para *django*

El servidor *HTTP Apache* es un servidor web HTTP de código abierto, para plataformas *Unix* (*BSD*, *GNU/Linux*, etc.), *Microsoft Windows*, *Macintosh* y otras, que implemente el protocolo HTTP/1.1 y la noción de sitio virtual.

Para la instalación de *Apache* en ubuntu (debian) 9.10 como servidor de páginas web hay que teclear en el terminal:

```
# sudo apt-get install Apache2
```

A continuación se procede a la instalación de los módulos necesarios para que *Apache* pueda cargar código *python* al arrancar y se comunique con las librerias *django - python*. Llegados a este punto se puede decantar por dos tipos de modulos a instalar.

- *mod_wsgi*
- *mod_python*

Para éste proyecto se ha decantado por el primero de los procedimientos. Una explicación de por qué, es que el *mod_wsgi* usa menos memoria y está preparado para sitios con mayores exigencias. En la figura 2.1 se observa la arquitectura utilizada y la ubicación del módulo como comunicador entre el servidor *Apache* y el servidor *django*:

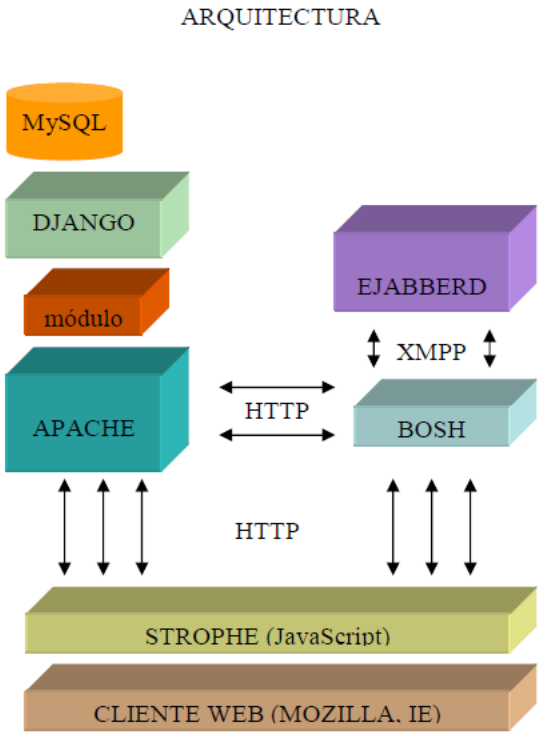

Figura 2.1: Arquitectura Apache <-> módulo <-> Django

mod_wsgi es similar a *mod_perl*, hace que se pueda utilizar código *python* en *Apache* y cargar código *python* en memoria cuando el servidor arranca. El codigo permanece en memoria según el ciclo de vida de un proceso en el servidor *Apache*, esto hace aumentar el rendimiento frente a otro tipo de servidores. *Django* funcionará con cualquier version de *Apache* que soporte *mod_wsgi*.
Para la instalación de éste módulo se teclea en el terminal:

```
# sudo apt-get install libApache2-mod-wsgi
```

mod_python La alternativa es utilizar el antiguo *mod_python*. Es el predecesor del módulo *mod_wsgi* pero funciona perfectamente y con grandes resultados.

Para la instalación de éste módulo se teclea en el terminal:

```
# sudo apt-get install libApache2-mod-python
```

2.2.3.1. Configuración básica de *Apache*: Usando el módulo *mod_wsgi*

Para que la explicación sea más clara, con el comando *ls -l* se va a listar los ficheros y carpetas que contiene la ruta */etc/apache*2:

```
# /etc/Apache2$ ls -l
total 80
-rw-r--r-- 1 root root  8097 2010-11-18 22:16 Apache2.conf
drwxr-xr-x 2 root root  4096 2011-02-08 12:40 conf.d
-rw-r--r-- 1 root root   551 2010-11-18 22:16 envvars
-rw-r--r-- 1 root root   125 2011-03-10 13:09 httpd.conf
-rw-r--r-- 1 root root 31063 2010-11-18 22:16 magic
drwxr-xr-x 2 root root  4096 2011-03-10 13:07 mods-available
drwxr-xr-x 2 root root  4096 2011-03-10 13:06 mods-enabled
-rw-r--r-- 1 root root   513 2011-03-09 10:49 ports.conf
-rw-r--r-- 1 root root   513 2011-03-09 09:53 ports.conf.backup
drwxr-xr-x 2 root root  4096 2011-03-09 14:24 sites-available
drwxr-xr-x 2 root root  4096 2011-03-09 13:35 sites-enabled
```

Una vez instalado el *mod_wsgi*, como se ha comentado en el paso anterior, se irá a la carpeta *sites-available*, en ella se va a crear un nuevo *site* (se crea un nuevo fichero llamado *django*-chat*) dentro* de *sites-available*. El path típico de instalación de *Apache* para éste nuevo *site es:*

```
# /etc/Apache2/sites-available/
```

En esta ruta se crea el nuevo fichero (*site*), llamado *django-chat*. El código a escribir es el siguiente:

```
<Location "/chat">
        SetHandler python-program
        PythonHandler django.core.handlers.modpython
        PythonDebug On
        PythonPath
         "['/home/miguel/Documentos/djcode/django_chat/chat/',
         '/home/miguel/Documentos/djcode/django_chat/dojango/',
         '/home/miguel/Documentos/djcode/django_chat/',
         '/home/miguel/Documentos/djcode'] + sys.path"
        SetEnv django_SETTINGS_MODULE django_chat.settings
</Location>
```

Y se activa el *site*:

```
# sudo ln -s
        /etc/Apache2/sites-available/django-chat
        /etc/Apache2/sites-enabled/django-chat
```

Se va a comentar las siguientes líneas:

```
.....
        PythonPath
         "['/home/miguel/Documentos/djcode/django_chat/chat/',
         '/home/miguel/Documentos/djcode/django_chat/dojango/',
         '/home/miguel/Documentos/djcode/django_chat/',
         '/home/miguel/Documentos/djcode'] + sys.path"
.....
```

Con esto se ha logrado que, cuando se escriba en el navegador *http://localhost/chat* se abra la página correspondiente al servicio de mensajería. L*as aplicaciones django* que el programador escriba necesita de un servidor que las sirva, como si de un servidor web se tratara. *django* facilita este servicio, como *framework* que es, con un servidor ligero que trae cuando se instala *django*. Hay que aclarar que el servicio de mensajería funciona tanto en modo desarrollo, es decir, con el servidor que ofrece *django*, como un servidor de páginas web como es *Apache*. Se ha decantado por *Apache* por ser un servidor muy utilizado hoy en día y de código abierto.

En P*ythonPath* se escribe la ruta de las *aplicaciones django*, en este caso *chat* y *dojango*. *djcode* es una carpeta que contiene proyectos django, el proyecto *django_chat* contiene a su vez *aplicaciones django:*

chat aplicación servidora de chat.

dojango aplicación cuya función es de apoyo, en la comunicación servidor - cliente. Sobre ésta aplicación hablaremos con más detenimiento en la sección 2.4.

A continuación se va a explicar la instalación del módulo *mod_python*.

2.2.3.2. Configuración básica de *Apache*: Usando el módulo *mod_python*

Para esta configuración se crea, al igual que el anterior, un nuevo *site*. Se edita el archivo *default* que viene por defecto en la instalación de *Apache*:

```
# nano /etc/Apache2/sites-available/default
```

En este archivo se va a especificar un nuevo *VirtualHost*, se determina que se usará un manejardor de *python* para servir las *aplicaciones django*. Además con la variable de entorno *Pythonpath* se determina el *path* a las *aplicaciones django* y el *path* al código fuente de django, como muestra el siguiente código:

```
<VirtualHost 0.0.0.0:80>

        ServerName name.of.de.server
        ErrorLog "/tmp/error_log"
        CustomLog "/tmp/access_log" common
        <Location "/">
                SetHandler python-program
                PythonHandler django.core.handlers.modpython
                SetEnv django_SETTINGS_MODULE mysite.settings
                PythonPath
                 "['/path_a_tu_aplicacion/',
                 '/path_to_django_installation/django/src'] +
                 sys.path"
                PythonOption django.root /
                PythonDebug On
        </Location>
```

```
[..]

</VirtualHost>
```

PythonOption django.root / es una opción nueva en *django 1.x*. Si se configurara *PythonOption django.root /mysite*, *django* añadiría automáticamente */mysite* a todas las urls de la aplicación. Así puedes añadir mas *sites* bajo otras urls, por ejemplo añadir /mysite2, para ello solo se tendrá que cambiar la opción *django.root*.

Es mejor añadir la opción *PythonPath* aquí aunque se puede quitar si la variable de entorno *PythonPath* del sistema se encuentra ya correctamente configurada. Como se puede observar también se introduce en el path la variable *sys.path* para importar otras librerías y hacer uso de ellas. Para temas de rendimiento también se pueden añadir otras directivas como *PythonAutoReload Off*. Para más información y documentación del *mod_python*:

http://modpython.org/live/current/doc-html/directives.html

2.2.4. Instalación de las aplicaciones *chat, dojango* en *django*

Hoy en día en internet hay multitud de programas chat comerciales o gratuitas, cada una de ellas con sus pros y sus contras.

Se suele pensar que las aplicaciones libres o de código abierto, comúnmente, ofrecen problemas de seguridad y, un plazo corto de soporte gratuito, ademas si el programa es poco utilizado, el desarrollador abandona el proyecto. No es así, los programas de código abierto son muchas veces productivos y favorables, por varias razones que a continuación se van a comentar.

Los programadores pueden efectuar cambios en el código libre[4], de forma que, puedan modificar código para que realicen unos objetivos u otros según sus pretensiones, de forma que en un tiempo corto, permita conseguir la realización de un proyecto.

Los comentarios anteriores hacen alusión a lo que se va a comentar a continuación. Para la realización del proyecto se ha investigado en internet información relacionada con el mundo del chat con *django*. Un autor de código libre diseñó una aplicación *chat* en *django*. Los pasos siguientes explican la instalación de dicha *aplicación django*

[4]Se tiene que tener también cuidado con el tipo de licencia ofrecida por el autor de dicho código libre. Existen multitud de aplicaciones libres con diferentes licencias, algunas ofrecen una completa liberación del mismo de forma que se puede modificar la totalidad del código además con la posibilidad de comerciar con él. Otros programadores en cambio ofrecen el código pero sin una comercialización del mismo.

de código abierto que ofrece por una parte al cliente, una interfaz web con la que interactuar, chatear y comunicarse con otros usuarios y un servidor que proporciona el manejo del intercambio de información de un usuario a otro.

Hay de aclarar que esta aplicación *es gratuita y de código libre*. Nuestro proyecto está ligado a esta aplicación. Más adelante se explicará grandes cambios efectuados en el código, ya que, lo básico está escrito pero hay bastantes cambios para llegar al objetivo final de este proyecto. Es decir, por ejemplo, la creación de canales, usuario/password, posibilidad de comunicación con otras aplicaciones *django* como videollamada, etc.

- Primeramente se descarga la *aplicación chat* de la siguiente dirección:
  - http://code.google.com/p/django-chat/downloads/detail?name=django-chat-0.2.zip
  - También se puede encontrar en el CD suplemantario en la carpeta *software*, *django-chat-0.2.zip*
- Y también la *aplicación dojongo*:
  - http://dojango.googlecode.com/files/dojango-0.3.tar.gz
  - También se puede encontrar en el CD suplementario en la carpeta *software*, dojango-0.3.tar.gz

2.2.4.1. Descarga de las aplicaciones

Una vez que las aplicaciones se han descargado, se descomprimen. Importante: Renombramos la carpeta descomprimida de *django-chat-0.2* a *django_chat*. Se recalca que estos pasos son diferentes a los que el autor proporciona en la página web oficial.

```
# unzip django-chat-0.2.zip -d django_chat
# tar xvzf dojango-0.3.tar.gz
# mkdir djcode
# mv django_chat ./djcode/
# mv dojango ./djcode/django_chat/
```

2.2.4.2. Configuración de la aplicación *chat*: fichero */django_chat/settings.py*

Según la guía de instalación de la página web del desarrollador se pide crear una base de datos. Por defecto en el archivo *setting.py* viene preconfigurada la base de

datos sqlite(*sql3*). Para este proyecto, la base de datos que utilizaremos será *My*SQL. *MySQL,* ademas de ser software libre, es un sistema gestor de bases de datos relacional, multihilo y multiusuario con más de seis millones de instalaciones.

Teniendo instalado *MySQL*[5] se crea una base de datos y se configura los parámetros del archivo */django_chat/settings.py*:

```
DATABASE_ENGINE = 'mysql'           # 'postgresql_psycopg2', 'postgresql', 'mysql',
                                    'sqlite3' or 'oracle'.
DATABASE_NAME = 'chatdb'            # Or path to database file if using sqlite3.
DATABASE_USER = 'admin'             # Not used with sqlite3.
DATABASE_PASSWORD = 'chat'          # Not used with sqlite3.
DATABASE_HOST = ''                  # Set to empty string for localhost. Not used with
                                    sqlite3.
DATABASE_PORT = '3306'              # Set to empty string for default. Not used with
                                    sqlite3.
```

Vamos a aclarar las líneas del código anterior:

- *Database_Engine = 'mysql'*
  - Escogemos entre las opciones el sistema de base de datos a utilizar, en este caso *MySQL.*
- *Database_Name = 'chatdb'*
  - Nombre de la base de datos utilizada. Debe haber sido creado por el administrador de la base de datos, hay que asignar también los permisos correspondientes.
- *Database_User = 'admin'*
  - Usuario de la base de datos *chatdb*, este usuario tiene los permisos que se la haya dado al crear la base de datos *chatdb.*[6]
- *Database_Password = 'chat'*
  - Al crear una base de datos para poder conectarse a ella hay que indicar el *password.*

[5]La instalación de MySQL no es objetivo de este proyecto, se remite al lector buscar guias de instalación de MySQL (si es el sistema servidor de base de datos que quiere utilizar) en la página oficial http://www.mysql.com/.

[6]Durante el desarrollo del proyecto se ha utilizado como apoyo, para la creación de la base de datos y la asignación de privilegios (permisos), la herramienta *MySQL Administrator.* Otra herramientra, muy utilizada, para ver el contenido de una base de datos es el programa *MySQL Query Browser.*

- *Database_Host = ”*
  - No se ha dado ningun nombre ni dirección ip, ya que, si lo dejamos en blanco se refiere a *localhost*. *My*SQL se alojó en la misma máquina donde se instaló *django*.
- *Database_Port = ’3306’*
  - Puerto de conexión a la base de datos *MySQL*, que por defecto es *3306*.

2.2.4.3. Configuración de la base de datos: sincronización *MySQL* - *django*

Una vez creada la base de datos y configurado el archivo */django_chat/settings.py* se debe sincronizar, esta tarea se lleva a cabo con el siguiente comando.[7]

Importante: se debe estar situado bajo la carpeta *django_chat* :

```
# python manage.py syncdb --noinput
```

- El superusuario debe ser *admin*[8]
- Y el password *chat*[9]

2.2.4.4. Prueba de las aplicaciones: Inicio del chat

Para poner en marcha la *aplicación chat* con el servidor de desarrollo *django* se ejecuta el siguiente comando bajo la carpeta *django_chat*:

```
# python manage.py runserver
```

Se puede especificar otro PUERTO diferente en el que el servidor se pone a la escucha de peticiones del cliente. El puerto por defecto es el 8000. Para iniciar en otro PUERTO se hace con el comando siguiente:

[7] Este paso es característico en *django*. Se remite al lector consultar la bibliografia para más información.
[8] Usuario de la base de datos *chatdb*
[9] Contraseña de la base de datos *chatdb*

```
# python manage.py runserver PUERTO
```

Accediendo a la url http://localhost:8000/chat se accede a la interfaz web cliente de mensajería, ver figura 2.2:

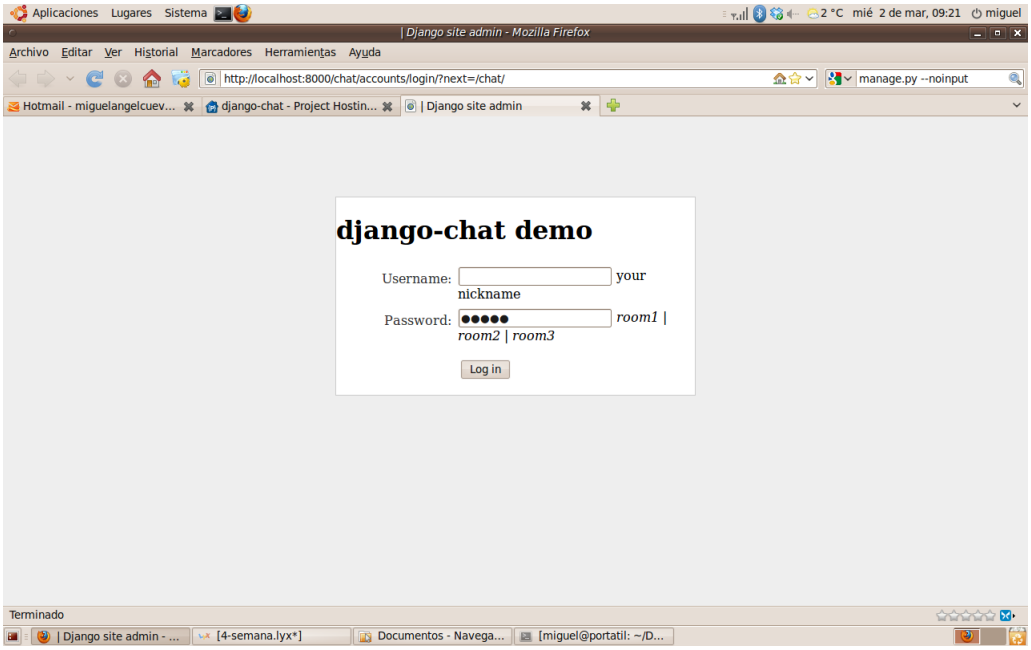

Figura 2.2: Test Page

Ésta es la página de la aplicación web *chat* del desarrollador del programa de mensajería. Para acceder a la aplicación con introducir cualquier nombre de usuario se le estará permitiendo tener acceso al chat y una conexión a un *room* o *canal*. Observación importante: no hay ningún tipo de impedimento en la entrada de un usuario al chat. La aplicación es pública. Como se irá indicando en los siguientes apartados, se iran produciéndose grandes cambios para adaptarlos al objetivo final del proyecto. Un adelanto sería, la adaptación a usuarios con privilegios registrados en la base de datos por el administrador. Se posibilita en el proyecto un usuario público, con la restricción de conectarse a un solo canal por defecto, imposibilitando el acceso a otros canales.

En la figura 2.3, en líneas generales, se observa que, la parte derecha se situa la herramienda chat mediante el cual se posibilita la comunicación con otros usuarios y en la parte izquierda información acerca del programa:

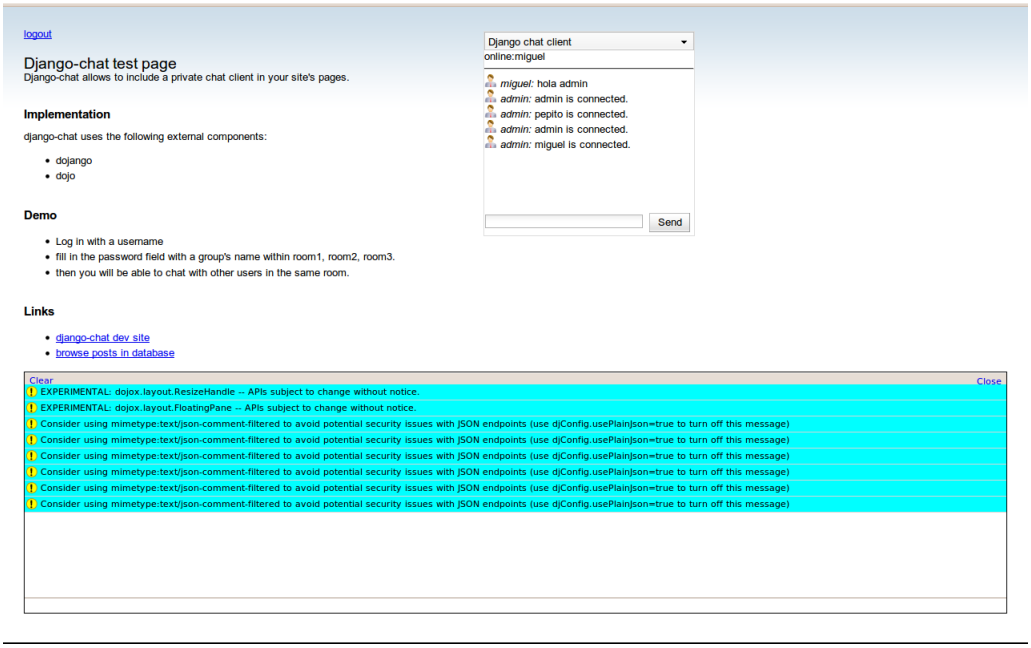

Figura 2.3: Chat en django

2.3. Tipo de Licencia de la *aplicación chat*

Tiene licencia *BSD*, es la licencia de software otorgada principalmente para los sistemas *BSD* (*Berkeley Software Distribution*). Es una licencia de *software libre permisiva* (una licencia de software libre permisiva, también conocida como minimalista o liberal, es una licencia de software libre flexible respecto a la distribución, de modo que el software pueda ser redistribuido como software libre o software propietario, siendo libre la licencia original del autor), como la licencia de *OpenSSL* o la *MIT License*. Esta licencia tiene menos restricciones en comparación con otras como la *GPL* estando muy cercana al dominio público. La licencia *BSD* al contrario que la *GPL* permite el uso del código fuente en software no libre.

2.4. Información sobre la aplicación *dojango & dojo*

La idea principal es la siguiente. El proyecto de mensajería se compone principalmente de dos aplicaciones:

- *Aplicación chat*
- *Aplicación dojango*

La principal característica de la aplicación *chat* es el uso de las herramientas que le proporciona la aplicación *dojango*. *dojango* es una aplicación *django* reutilizable que actua sobre el lado cliente dentro del proyecto django. Es decir, en el paradigma cliente-servidor, el cliente es el navegador con *JavaScript* implementado. El servidor es la aplicación *chat* que espera peticiones por parte de los clientes. Un ejemplo claro es la utilización de la variable *wait value*, que informa al cliente cada cuanto tiempo tiene que refrescar su página para actualizar la aplicación chat. La página web enviada por el servidor */django_chat/chat/templates/test.html* se observa la variable antes comentada.

```
.........
<script type="text/javascript">
        // dojo.requires()
        dojo.require("dojox.layout.FloatingPane");
        dojo.require("dijit.form.TextBox");
        dojo.require("dijit.form.Button");
        dojo.require("dojo.parser");
        // scan page for widgets
        // scan page for widgets and instantiate them
        dojo.require("dojo.parser");

        function chat_update() {
                var wait = 30000  <--------------------------------
                dojo.xhrGet({
                  url: "./update",
                  handleAs: "json",
                  load: function(data, ioargs) {
                   dojo.byId("messages").innerHTML = data.users +
                                '<hr>' + data.messages;
                   wait = data.wait_value * 1000
                   setimeout(chat_update, wait)
.........
```

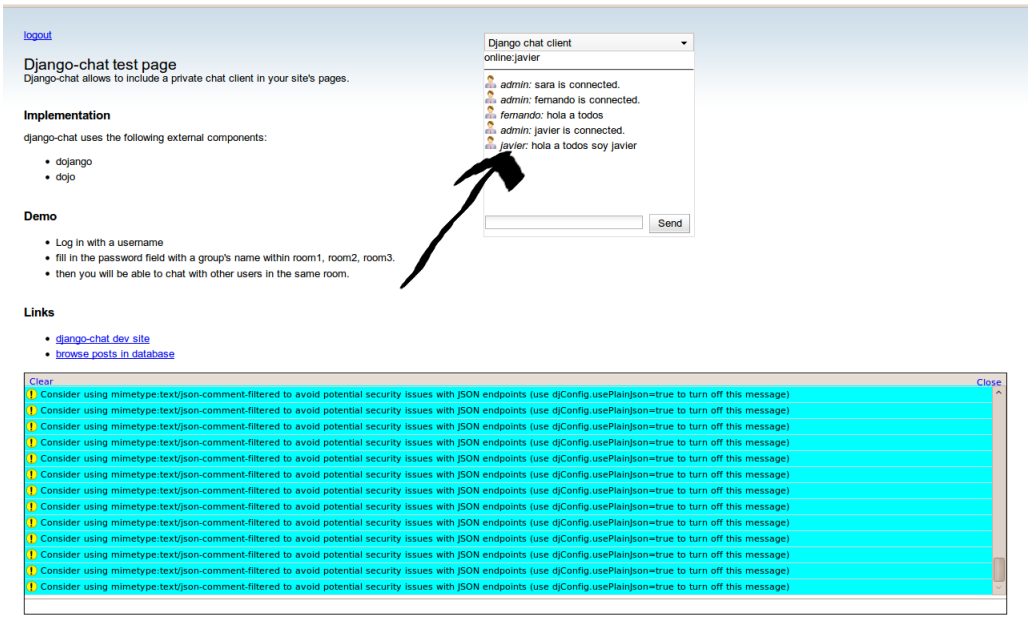

Figura 2.4: Página principal del chat

Este script actualiza el *frame* de la figura 2.4 (marcada con flecha <–):

3 Estudio de las aplicaciones *django* con *Wireshark*

3.1. Estudio con *WireShark*

Wireshark es un analizador de protocolos utilizado en el análisis y solución de problemas en redes de comunicaciones, desarrollo de software y protocolos.

Ya se ha visto en el capítulo 2 la instalación y configuración de los programas utilizados, con este analizador se capturó los mensajes[1] que intercambiaban cliente-servidor para obtener un análisis de las comunicaciones. Éste análisis se obtuvo debido a los errores que se producían cuando se intentaba descargar las páginas web que sirve el servidor de django que viene por defecto, por ejemplo, no se descargaban las hojas de estilos .css, imágenes .jpg, etc, y es por ello el estudio y la solución a los errores de instalación.

La aplicación usó filtros para capturar solo el protocolo http. La información capturada se guardó en ficheros. Mediante el CD suplementario el lector puede descargar y visualizar las capturas que se originó. Dichos ficheros estan alojados en la carpeta *capturas-wireshark*. Nota importante: en el capítulo 4 se habla de modificaciones que se le van a aplicar a la aplicación chat, de lo que se habla en éste capítulo se refiere a la aplicación chat originaría, es decir, el del autor de la aplicación chat. El estudio se compone de tres fases:

- *1-acceso-principal* . Archivo .pcap para wireshark, en este archivo se encuentra la captura del acceso a la página principal al chat. Para más información ver apéndice A.1 para su análisis en detalle.
    - CD://capturas-wireshark/1-acceso-principal
- *2-logueo y chateo*. Archivo .pcap para wireshark donde se encuentra la conversación entre cliente, envío de usuario (ejemplo: *javier*) y canal (ejemplo: *rooma*),

[1] Se verá que a estos mensajes se le han denominado conversaciones también.

y la respuesta por parte del servidor permitiendo el acceso al mismo. Este servidor permite la entrada a cualquier canal con cualquier nombre de usuario, siempre y cuando se teclee de forma correcta el canal al cual quiere conectarse (*rooma, roomb* o *roomc*), en caso contrario, no permite seguir. En los siguientes capítulos se habla de las modificaciones que se realizaron en el acceso a la interfaz web del cliente chat, es decir, la página web de inicio se ha modificado completamente. Una vez dentro, se puede comenzar a chatear con los demas usuarios conectados al mismo canal. Para más información ver apéndice A.2 para su análisis en detalle.

  - CD://capturas-wireshark/2-logueo

- *3-desconectar* . Archivo .pcap para wireshark que captura la conversación http para la finalización de la aplicación chat. Para más información ver apéndice A.3.

  - CD://capturas-wireshark/3-desconectar

4 Modificaciones en la aplicación *django-chat*

4.1. Introducción

Los capítulos anteriores hablaban sobre los conocimientos adquiridos al inicio del desarrollo del proyecto y la instalación de los programas necesarios. El tiempo dedicado a ello fue largo y monótono pero es un trabajo que había que dedicarle. En casi todo los proyectos finales de carrera el inicio es el más duro pero una vez desarrollado este trabajo, la situación se encauza y se hace maś ameno.

Éste capítulo trata sobre las modificaciones necesarias que se le ha tenido que adaptar a la *aplicación chat* para llegar al objetivo final. Se recuerda que el objetivo final es la comunicación entre un cliente que utiliza un navegador web con *javascript* y un servidor *Apache* que a su vez interactúa con el servidor de aplicaciones *django.* Pero para llegar a este objetivo hay modificaciones importantes en el código. Estas modificaciones se detallan en los siguientes apartados. Se va a resumir en los siguientes párrafos las modificaciones realizadas.

Primeramente se restringió los usuarios que accedían a la *aplicación chat*, usuarios registrados en la base de datos y usuarios invitados con menos privilegios sin necesidad de registro, y es por tanto, el requisito de un password para los usuarios registrados. Esta forma de entrada no se proporcionaba, si no que por el contrario, el password según el autor era el canal al cual se quería acceder, se ve que es un cambio bastante abrupto. En la figura 4.1 se tiene un resumen de las nuevas características.

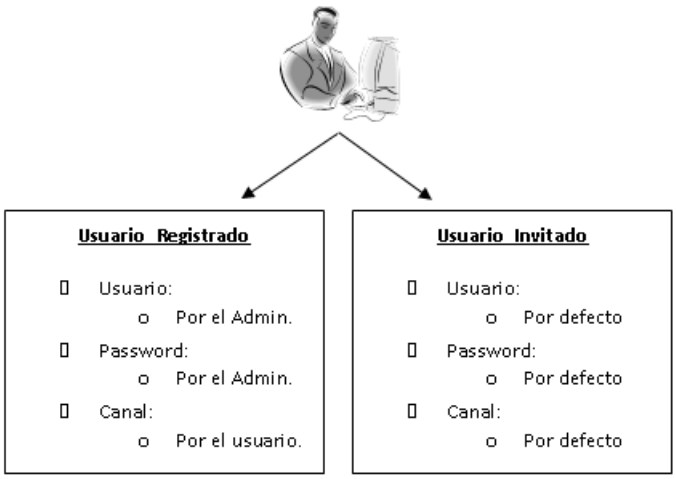

Figura 4.1: Diferencia entre usuario registrado y usuario invitado

Seguidamente se adaptó la *aplicación chat*, también se va a llamar módulo chat, para que interactuara con otros módulos (o aplicaciones), más directamente con el módulo videollamada. Ésta es la única razón por la que no se trabajó con los programas que se mencionan en el capítulo 5, ya que, estos programas trabajan con la estandarización como por ejemplo XMPP y se necesitaba alguna forma de comunicación entre usuarios para iniciar o detener una videollamada.

Lo siguiente fue otra mejora a la *aplicación chat*, y más que mejora sería una rectificación completa del uso de la red. La *aplicación chat* trabajaba de la siguiente forma:

- Se almacenaba lo escrito por un usuario en la base de datos, y se reenviaba todo a los demás usuarios y a él mismo.
- El usuario que había escrito lo anterior cada cierto tiempo refrescaba la página web y el servidor reenviaba nuevamente lo que había escrito, malgastando el uso de la red. Pero el problema era cuando el chat iba incrementándose tanto en usuarios como en conversaciones.

Nuevamente la solución que se adoptó fue una comunicación no estandarizada entre usuario y servidor de forma que remita solo las conversaciones nuevas y no todo, así como una modificación en la página web utilizando *javascript*.

Por último las últimas modificaciones son algunas características de las que se habla en el capítulo 1 y que debe tener todo cliente chat, como mínimo.

4.2. Modificaciones en la aplicación *chat*

4.2.1. Página principal de registro de usuario

La página de registro de usuario que se observa en la figura 4.2 es completamente diferente al proporcionado por el desarrollador.

Se diferencian dos tipos de usario:

usuario Usuario registrado en la base de datos (*MySQL*). Previamente agregado por el administrador del servidor chat utilizando la página de administración de *django*. Posibilidad de utilizar el protocolo *ldap,* ver sección 4.3.

invitado Usuario no registrado en la base de datos. Puede hacer uso del programa chat. La diferencia más significativa es el nombre con el que entra al programa chat. El servidor realiza una serie de mecanismos para asignarle un nombre de usuario del tipo **invitado-XXX**. Donde **XXX** será una serie de letras en mayúscula y minúscula aleatorias. Con respecto al password, tampoco tendrá que teclear nada, el servidor le asignará una.

La página principal de registro se ha modificado para que los usuarios no registrados no puedan acceder al mismo, de manera que, para iniciar una sesión se necesita usuario y contraseña proporcionado por el administrador. Aún así, se puede acceder y disfrutar del programa (con menos privilegios) como *invitado*. Se ha de marcar la opcion *invitado*, la página web está configurado con un *javascript,* de forma que, una vez se halla marcado *invitado*, los campos usuario y password esten bloqueados rellenándolos por unos valores por defecto. Cómo ya se comentó, el canal por defecto del *invitado* serál el *canal 1*. No permitiendo el acceso a los demas.

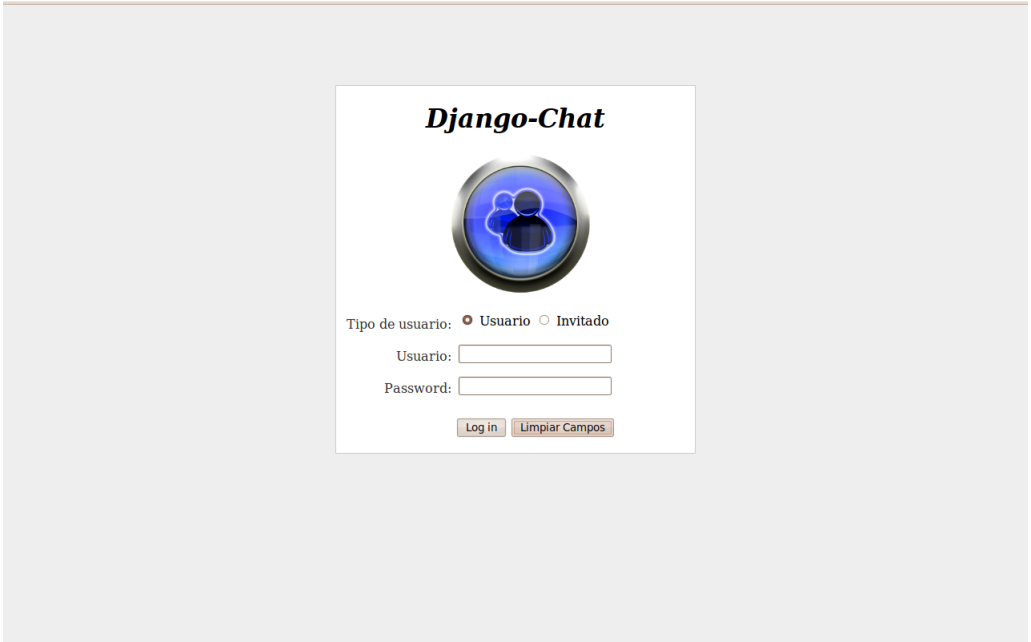

Figura 4.2: Página de registro de usuarios

Se ha sacado la elección del canal del registro de usuario, para selecionarlo en otra página web distinta. Se ha conseguido de esta forma no modificar código interno de django[1]. La siguiente imagen muestra la elección del canal al cual se quiere conectar el usuario.

[1]Si en la página web de registro de usuario (*usuario*, *password*) se añade otro campo, *canal*, hay que modificar código interno de *django*, es decir, ficheros de configuración de *django*. Por ello se ha escogido otro camino diferente para la misma finalidad. Este proyecto se ha escrito de manera que, el código sea *portable* a otros sistemas que esten en funcionamiento y que no alteren el mismo.

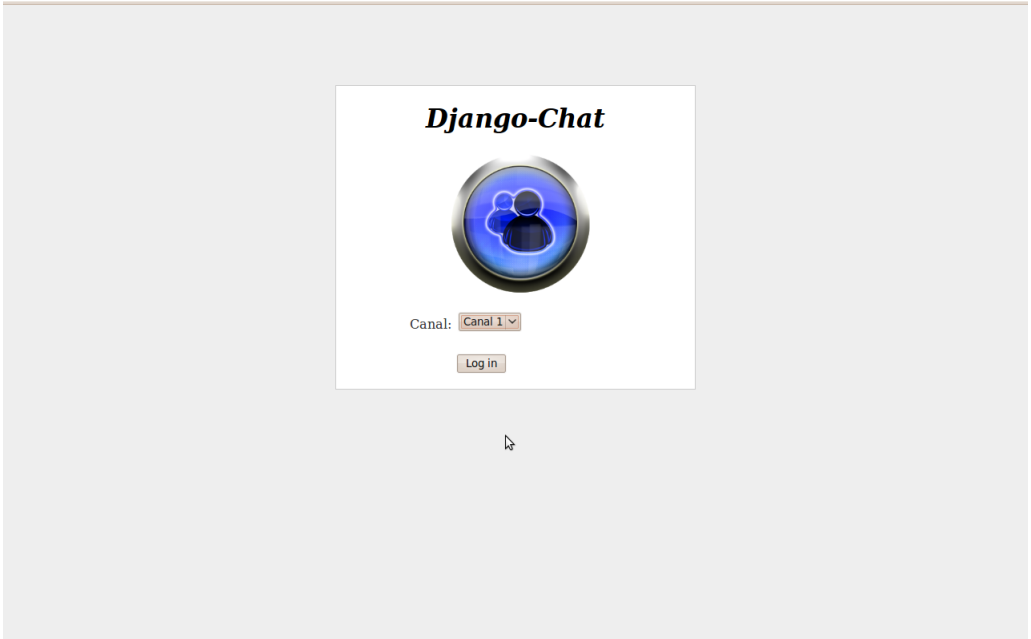

Figura 4.3: Elección del canal

Se ha modificado código del proporcionado por el desarrollador para contemplar estos aspectos. A continuación, se detallan las modificaciones que se han llevado a cabo.

4.2.2. Modificación de ficheros

4.2.2.1. Fichero *../django_chat/backends/passgroup.py*

1. Se quería conseguir tener una secuencia de invitados para definir el nombre con el que entraba al chat, en vez de la serie de letras **XXX** que hemos adoptado finalmente. Leyendo manuales de python recomiendan no crear variables globales (evitar a toda costa).

2. El segundo aspecto es un poco más difícil de explicar. Para ser breve, se han incluido nuevos campos en el modelo de *UserData* del fichero de modelos *../django_chat/chat/models.py* .

```
class UserData(models.Model):
    """
    """
    user             = models.ForeignKey(User, unique=True)
    last_req         = models.DateTimeField(default=datetime.now())
    usecond_last_req = models.IntegerField(default=0)
    refresh_period   = models.IntegerField(default=5)
    connected        = models.BooleanField(default=False)
    disconnected     = models.BooleanField(default=False)
    remote_user      = models.CharField(default="inactivo",
                                        max_length=255)
    status_call      = models.IntegerField(default=0)
    canal            = models.IntegerField(default=1)
    mensaje_privado  = models.CharField(default="null",
                                         max_length=255)
    flag_message     = models.IntegerField(default=1)
    lock             = models.IntegerField(default=0)
    def avatar_url(self):
         return 'http://goflow.alwaysdata.net/images/user16.png'
    class Meta:
         verbose_name        = 'User data'
         verbose_name_plural = 'Users data'
```

Donde:

user: usuario registrado en la BBDD por el administrador.

last_req: fecha con la hora de la última solicitud. Traspasado un tiempo y si el cliente no da síntomas de estar conectado, se le desconectará.

refresh_period: periodo de refresco de actualización del cliente chat.

disconnected: de tipo booleano que indica si se ha desconectado desde alguna máquina donde inició sesión.

canal: canal del usuario al cual está conectado.

Hay que comentar que un usuario registrado puede loguearse en más de un equipo, las conversaciones que irá teniendo aparecerá en todos los equipos donde tenga abierta la página chat. Se ha cambiado el cierre de la página de forma que, podamos abrir en distintas máquinas el mismo usuario, pero si cerramos una, inmediantamente se

cerraran en todas. Se ha realizado de esta forma por seguridad. Por ejemplo, si se está en casa chateando con una persona, e inmediatamente se deja la conversación y se sigue la conversación en otro lugar por ejemplo en la facultad, pueda abrir el chat con la conversación que dejó antes de irse. Si se le olvidará cerrar el chat en la facultad, puede cerrarlo desde casa, cerrándo éste. Para más información ver la sebcción 4.2.2.4.

4.2.2.2. Template: *../django_chat/chat/templates/login.html*

Como se vió en la figura 4.2 se ha cambiado el estilo y la forma con la que se recogen los datos introducidos por el usuario. Distinguiendo entre usuario registrado y usuario invitado. Características a comentar:

- Se ha usado la tecnología *JavaScript* para ocultar los campos cuando pinchamos en *invitado*.
- Se ha agregado un campo de reseteo en el formulario, *Limpiar campos*.

Estas son las nuevas líneas de código empleadas:

```
"""
<div id="content-main">
 <form action="{{ app_path }}"  method="post" id="login-form"
                                name="formulario">
   <div class="form-row">
    <label for="id_tipo">{% trans 'Tipo de usuario:' %}</label>
    <input      type="radio"    name="tipo"     value="usuario"
                checked="checked"
                onclick="document.formulario.username.value = '';
                document.formulario.password.value = '';
                document.formulario.username.readOnly = false;
                document.formulario.password.readOnly = false"/> Usuario
    <input type="radio" name="tipo" value="invitado"
                onclick="document.formulario.password.value = 'aleatorio';
                document.formulario.username.value = 'invitado';
                document.formulario.username.readOnly = true;
                document.formulario.password.readOnly = true"/> Invitado
   </div>
   <div class="form-row">
    <label for="id_username">{% trans 'Usuario:' %}</label>
    <input type="text" name="username" id="id_username"/>
   </div>
   <div class="form-row">
    <label for="id_password">{% trans 'Password:' %}</label>
```

```
    <input type="password" name="password" id="id_password" value=""/>
   </div>
   <div class="submit-row">
    <input type="submit" value="{% trans 'Log in' %}"/>
    <input      type="reset" name="limpiar" value="Limpiar Campos"
                onclick="document.formulario.username.value = '';
                document.formulario.password.value = '';
                document.formulario.username.readOnly = false;
                document.formulario.password.readOnly = false" />
   </div>
 </form>
</div>
"""
```

4.2.2.3. Template: *../django_chat/chat/templates/test.html*

En la figura 4.4 se ve la página principal del chat. Se va a comentar las nuevas características para adaptarlos a nuestras necesidades.

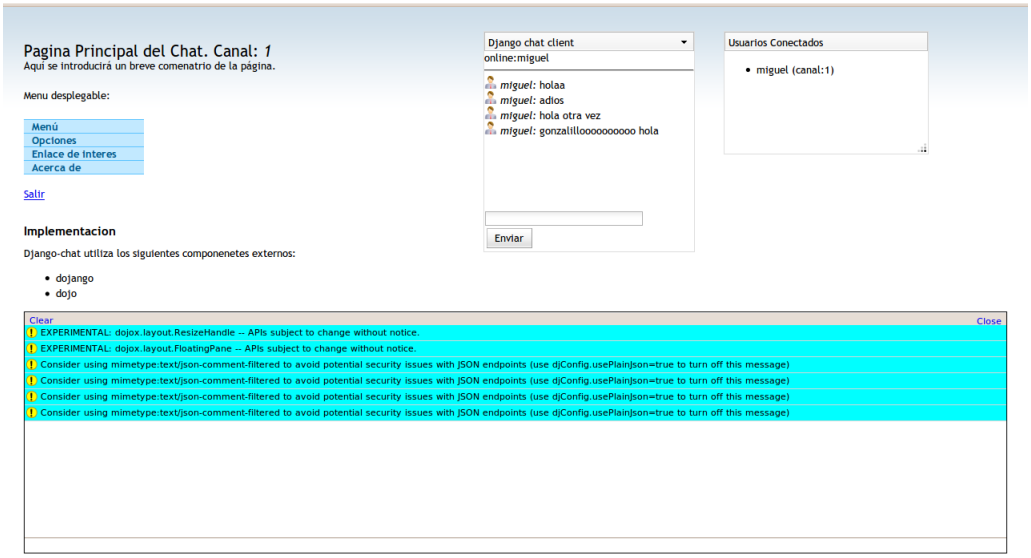

Figura 4.4: Página principal del chat

ventana:usuarios_conectados En esta ventana se observa los usuarios que han iniciado sesión en el chat independientemente en que canal se encuentren y además, se puede ver el canal al cual se han conectado, por ejemplo en la figura 4.4:

miguel (canal: 1)

menu_desplegable: Es un menú con submenus deslizantes, haciendo click en la opcion deseada produce la creación de una nueva ventana/pestaña. Es una creación propia escrita en *javascript*, ya que, al comenzar el estudio de django se estudiaron otros lenguajes de programación que no se sabían. Este código es fruto de dicho estudio.

4.2.2.4. Fichero *../django_chat/chat/views.py*

Se ha agregado una nueva función *logout(request)*:

```
def logout(request):
 usuariodesconectado = User.objects.get(username= request.user)
 udata, created = UserData.objects.get_or_create(user=
                                                  usuariodesconectado)
 udata.disconnected = True
 udata.save()
 usuariodesconectado.groups.clear()
 usuariodesconectado.save()
 auth.logout(request)
 return HttpResponseRedirect("/chat/")
```

El usuario cierra la conexión con el servidor, cerrando las sesiones que tenga abiertas en otras máquinas. Además se ha agregado la siguiente línea en el archivo *../django_chat/urls_prefix.py:*

```
(r'^admin/logout/$', 'django_chat.chat.views.logout'),
```

Con esta línea indicamos, que si nos pide la url, por ejemplo,
localhost:8000/chat/admin/logout/
ejecute la función definida en *django_chat.chat.views.logout* que es la función anteriormente descrita.

4.3. Uso de LDAP

Para utilizar LDAP en nuestra aplicación chat con Django, vamos a seguir la documentación que nos proporciona la siguiente dirección web:

http://packages.python.org/django-auth-ldap/

LDAP proporciona el "backends" o la autenticación en proyectos djangos en cualquier servidor LDAP. Para usarlo, se debe agregar la siguiente línea en el archivo de configuraciónn *settings.py:*

```
......
AUTHENTICATION_BACKENDS = (
 # 'django_auth_ldap.backend.LDAPBackend',
  'django_chat.backends.passgroup.GroupBackend',
 'django.contrib.auth.backends.ModelBackend',
)
......
```

hay que eliminar el # (que comenta la línea) para que se conecte con un servidor LDAP.

4.3.1. Autenticación básica. Configuración.

Si no tenemos un servidor LDAP local actualmente corriendo en la máquina en el puerto por defecto, tendremos que conectarnos al servidor añadiendo una línea en el archivo *setting.py*, tal como esta:

```
......
# AUTH_LDAP_SERVER_URI = "ldap://ldap.example.com"
......
```

Donde *ldap.example.com* es la *uri* con la que se conecta, también se puede dar la dirección ip (si es fija).

Una vez hecho esto, el primer paso es autenticar a un usuario con su contraseña contra el servicio LDAP. Hay dos maneras de hacer esto, *search/bind* o *bind*.

La primera consiste en conectar con el servidor LDAP ya sea de forma anónima o con una cuenta fija y buscando el nombre completo del usuario que se autentica.

Entonces se compara el nombre del usuario con la contraseña proporcionada por el usuario.

El segundo método consiste en obtener el DN del usuario de su nombre de usuario.

Debido a que las búsquedas LDAP aparecen en otras partes de la configuración, la clase *LDAPSearch* se proporciona para encapsular la información de búsqueda. En este caso, el parámetro de filtro debe contener el marcador de posición *% (user)s*. Una configuración sencilla para el primera forma *search/bind* sería el siguiente (incluidos algunos valores por defecto). Toda esta información va incluida en el archivo de configuración *settings.py*:

```
......
import ldap
from django_auth_ldap.config import LDAPSearch
......
AUTH_LDAP_BIND_DN = ""
AUTH_LDAP_BIND_PASSWORD = ""
AUTH_LDAP_USER_SEARCH = LDAPSearch("ou=users,dc=example,dc=com",
    ldap.SCOPE_SUBTREE, "(uid=%(user)s)")
......
```

Se realizará un enlace anónimo, busca coincidencia para "ou = user, dc = ejemplo, dc = com" en un objeto con un uid. La búsqueda debe devolver exactamente uno de los resultados o la autenticación fallará. Se tiene la posibilidad de buscar de forma anónima, se puede establecer *AUTH_LDAP_BIND_DN* con el nombre completo de un usuario autorizado y *AUTH_LDAP_BIND_PASSWORD* con la contraseña.

Para saltar la fase de búsqueda, se establece *AUTH_LDAP_USER_DN_TEMPLATE* a una plantilla ,el DN para autenticar al usuario directamente. Esta plantilla debe tener un marcador de posición *% (user)s*. Si el ejemplo anterior había utilizado *ldap.SCOPE_ONELEVEL*, lo siguiente sería un equivalente más sencillo (y eficiente):

```
......
AUTH_LDAP_USER_DN_TEMPLATE = "uid=%(user)s,ou=users,dc=example,dc=com"
......
```

Se tiene que tener en cuenta que todavía se debe establecer *AUTH_LDAP_BIND_DN* *AUTH_LDAP_BIND_PASSWORD* si se necesita algo más que una simple de au-

tenticación, como todas las operaciones de LDAP se llevará a cabo con estas credenciales. Esta es la coherencia y también porque de otro modo tendría que almacenar la contraseña del usuario en la sesión.

De forma predeterminada, las conexiones LDAP no son encriptadas y no hacen ningún intento por proteger la información confidencial, como contraseñas. Cuando se comunica con un servidor LDAP en el servidor local o en una red local, esto podría estar bien. Si se necesitara una conexión segura con el servidor LDAP, se puede utilizar un

ldaps: / / URL

o habilitar la extensión *STARTTLS*. Este último suele ser el mecanismo preferido. Para habilitar STARTTLS, se debe establecer:

```
......
AUTH_LDAP_START_TLS = True
......
```

4.3.2. Uso de objetos *User*

Autenticar contra una fuente externa es bueno, pero el módulo de autenticación de Django está estrechamente vinculado al modelo *django.contrib.auth.models.User*. Así, cuando un usuario inicia una sesión, tenemos que crear un objeto de usuario para que lo represente en la base de datos. Debido a que la búsqueda de un usuario en LDAP es sensible a los caracteres introducidos, la aplicación por defecto también busca los usuarios de Django con una consulta exacta y los nuevos usuarios se crean con nombres de usuario en minúsculas.

El único campo obligatorio para un usuario es el nombre de usuario, lo que obviamente se tiene. El modelo de usuario es exigente con los caracteres permitidos en nombres de usuario, por lo que LDAPBackend incluye un par de métodos, *ldap_to_django_username ()* y *django_to_ldap_username ()*, para traducir entre nombres de usuario LDAP y los nombres de usuario de Django.

4.3.3. Uso de LDAP. Instalaciones adicionales.

En la anterior subsección se ha comentado el uso de *LDAP* así como su instalación. A continuación se va a explicar otras características adicionales que no se han comentado y es fundamental, ya que sin estos no funcionaría.

4.3.3.1. Instalación de la librería python-ldap

Hay un par de librerías *LDAP* disponibles para *Python*, pero el más popular es el módulo de *Python-LDAP*, que (como con la *API* de *PHP*) utiliza la biblioteca *OpenLDAP C* como una base para proporcionar acceso a la red a un servidor *LDAP*.

OpenLDAP, así como la API de *Python-LDAP,* es de código abierto. Funciona en *Linux*, *Windows*, *Mac OS X*, *BSD* y otros sistemas *UNIX*. El código fuente está disponible en la página web oficial de *Python-LDAP*:

http://python-ldap.sourceforge.net

En este enlace se encuentra los binarios precompilados para varias plataformas, pero se va a instalar la versión desde el repositorio de Ubuntu.

Antes de instalar *Python-LDAP*, se debe tener el lenguaje de script Python instalado. Normalmente, este se instala por defecto en Ubuntu (y en la mayoría de Linux). La instalación de *python-ldap* requiere un único comando:

```
# sudo apt-get install python-ldap
```

4.3.3.2. Instalación de la librería python-django-auth-ldap

En django se tiene la posibilidad de autenticar a un usuario contra un servicio *LDAP*. Para la instalación se requiere un único comando:

```
# sudo apt-get install python-django-auth-ldap
```

Esta instalación requiere de las siguiente dependencias:

- *python* (>= 2.3)
- *python-django* : Entorno de trabajo de desarrollo web de alto nivel en *Python*
- *python-ldap* : Módulo de interfaz *LDAP* para *Python*, mencionado en la anterior subsección.
- *python-support* (>= 0.90.0) : Regeneración automática de los módulos *Python*

4.3.3.3. Corrección de fallo para usuarios invitados

Al cambiar la autenticidad para usar *LDAP* los usuarios invitados no tenían permisos para entrar al chat sin autenticarse. Este fallo ha sido solventado adoptando la siguiente medida.

Se recuerda que para autenticar a un usuario mediante *LDAP* introducimos en el archivo *settings.py* las siguientes líneas:

```
......
AUTHENTICATION_BACKENDS = (
        'django_auth_ldap.backend.LDAPBackend',
        'django.contrib.auth.backends.ModelBackend',
)
......
```

ahora se ha añadido una línea mas y el resultado es el siguiente:

```
......
AUTHENTICATION_BACKENDS = (
        'django_auth_ldap.backend.LDAPBackend',
        'django_chat.backends.passgroup.GroupBackend',
        'django.contrib.auth.backends.ModelBackend',
)
......
```

Se explica el funcionamiento: La primera línea conecta con el servidor *LDAP* para la autenticación del usuario. Si no tiene éxito intentará autenticarse con la nueva línea que se le ha dado, ésta utiliza la autenticación de django. Vamos a utilizar esta nueva autenticación para los usuarios invitados. Además el canal por defecto al que se conectarán es el canal 1, no se le está permitido conectarse a otro canal.

4.4. Integración con el módulo Videollamada. Modificación del programa chat.

4.4.1. Inclusión de una nueva ventana con la etiqueta html

Esta nueva ventana incluirá la pantalla de la videollamada. La ventana tiene dos estados:

- Estado desactivado/oculto: Estado por defecto, es el estado en el que se encuentra esta capa cuando no se realiza una videollamada.
- Estado activo/visible: Estado en el cual se inicia una videollamada, aparece una ventana con los fotogramas que captura la cámara web en el lado usuario (se ve en una ventana chica), y otra ventana superpuesta del usuario-llamante (se ve en una ventana más grande).

Hay alternancia entre un estado y otro, ya que se puede iniciar o finalizar una videollamada, punto que se verá a continuación.

4.4.2. Inicio y finalización de una videollamada.

En la ventana de usuarios conectados se agregó un icono junto al usuario como el de la figura 4.5:

Figura 4.5: Icono para iniciar una videollamada

Al hacer click en el icono que está junto al usuario remoto al que se quiere realizar una llamada, se ejecuta una serie de código JavaScript que modifica este icono,

siempre y cuando el usuario remoto acepte la llamada. El nuevo icono al que cambia es el de la figura 4.6:

Figura 4.6: Icono para concluir la videollamada

Con este nuevo icono se puede detener la videollamada.

4.4.2.1. Código JavaScript y Python para aplicar el punto anterior.

Interacción entre el código JavaScript del cliente y el código Python del servidor mediante variables. Estas variables son guardadas en las bases de datos, y cada usuario tendrá una. Las variables son definidas en el modelo como sigue:

```
......
remote_user = models.CharField(default="inactivo", max_length=255)
status_call = models.IntegerField(default=0)
......
```

Donde:

remote_user: guardará el nombre del usuario remoto, con el que realiza la videollamada. Es decir, Miguel tiene en la base de datos una variable *remote_user* = sara. Sara tendrá su variable *remote_user* = *miguel*. Esto nos sirve para ver con quién está hablando, además, si la variable *remote_user != inactivo (palabra reservada)*, cualquier otro usuario podrá realizar una videollamada sabiendo que no está hablando con nadie. Principalmente el objetivo es ese, que otro

usuario no pueda realizar una videollamada con otro usuario que esté ocupada realizando una. Los estados posibles son:

-1- Esperando confirmación por parte del usuario remoto

-2- Estado de la videollamada con exito, inicio de la videollamada

-3- Ocupado en una videollamada, si otro usuario solicita una videollamada a otro.

-4- Cancelación de una videollamada.

4.5. Mensajes privados entre usuarios.

Se ha dotado de una característica adicional a la aplicación. Esta característica es: el envío de un mensaje unidireccional entre un usuario a otro usuario.

En la figura 4.7 podemos ver como al hacer doble click en el usuario al que queremos enviar una mensaje se descubrirá una ventana donde podemos escribir lo que queremos enviar. Al hacer click en *"Click para enviar"* esta pantalla desaparecerá y comenzará el envío del mensaje al usuario correspondiente.

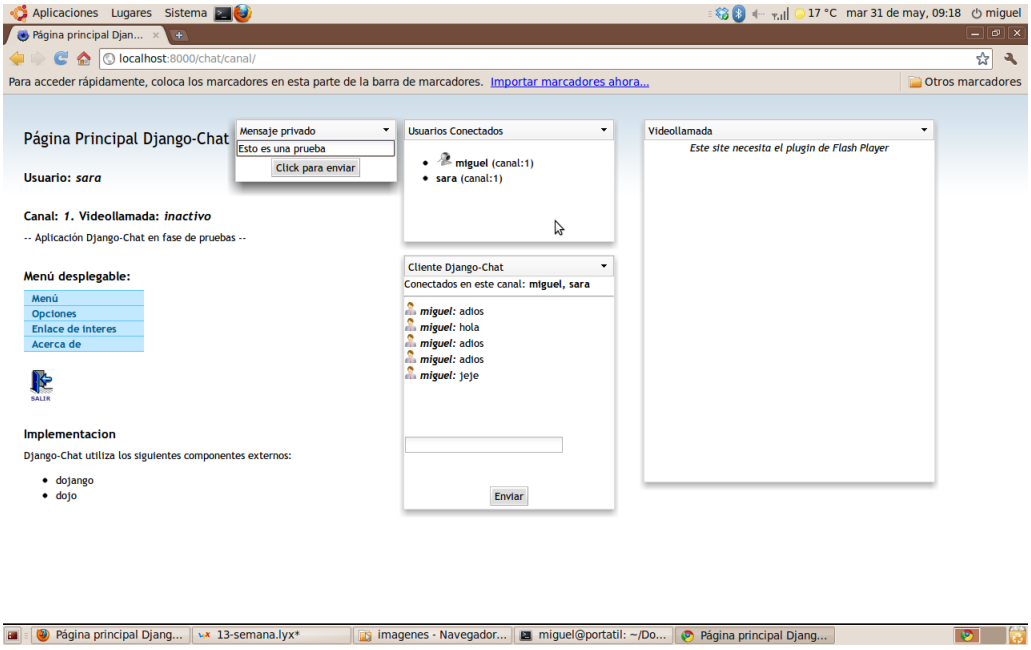

Figura 4.7: Mensajes privados entre usuarios

En la figura 4.8 se visualiza la llegada de un mensaje. Aparecerá una pantalla en la que aparezca el mensaje recibido por el otro usuario.

La notación seguida es la siguiente:

```
MP: (usuario) Mensaje a enviar
```

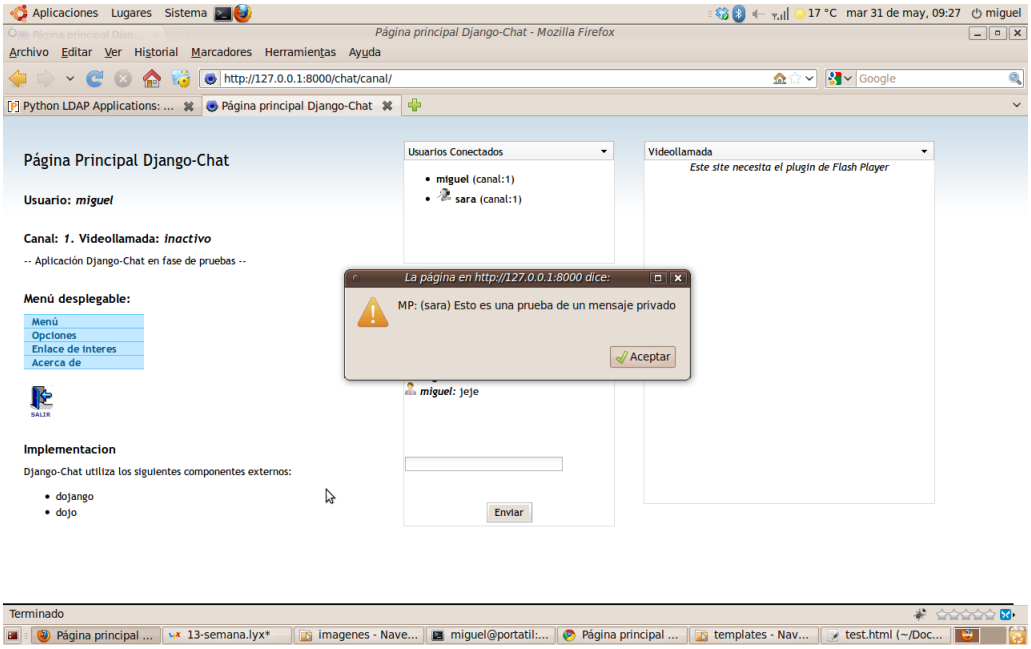

Figura 4.8: Llegada de un mensaje privado de un usuario

4.5.1. Mensajes privados. Código utilizado.

Para realizar la función explicada anteriormente, se ha utilizado código JavaScript en el cliente y código Python en el servidor.

4.5.1.1. Aparición/Desaparición de la ventana de edición del mensaje privado.

La pantalla de edición del mensaje estará escondida para el usuario a menos que el usuario haga doble click con el ratón en uno de los usuarios conectados, independientemente en que canal esté conectado. Con JavaScript desempeñaremos esta función de aparición y desaparición de dicha ventana.

Una vez rellenado el cuadro de texto a enviar se hace click en el botón *"Click para enviar"*. Se enviará un POST al servidor con los datos siguientes:

- Mensaje privado: texto plano cuyo contenido es el mensaje privado que se quiere enviar al otro usuario.
- Usuario remoto: usuario al que se le quiere enviar el mensaje.

Una vez clickeado desaparecerá la ventana.

4.5.1.2. Tareas del servidor.

Una vez llegado al servidor el archivo *views.py* detectará en el campo *text* que es un mensaje privado. ¿Cómo?, se ha explicado más arriba cómo se va a enviar el mensaje privado, es decir, de la siguiente manera:

```
MP: (usuario) Mensaje a enviar
```

Con la función *texto.find("MP:") >= 0* detectaremos que es un mensaje privado. A continuación se realizará las acciones oportunas para el envío del mensaje.

4.5.1.3. Visualización del mensaje al usuario.

El JavaScript del cliente, cuando el servidor le envía datos, determina si ha habido un mensaje privado para él, si lo detecta, lo muestra y empieza el proceso de reseteo del campo *mensaje privado* en el servidor para que puedan enviarle más mensajes privados. Este campo *mensaje privado* lo tienen todos los usuarios, que por defecto es nulo. Para una correcta comprensión se detalla a continuación los campos que se guardan para cada usuario en la base de datos, como modelo:

```
......
class UserData(models.Model):
"""
"""
    user = models.ForeignKey(User, unique=True)
        last_req = models.DateTimeField(default=datetime.now())
        refresh_period = models.IntegerField(default=5)
        connected = models.BooleanField(default=False)
        disconnected = models.BooleanField(default=False)
```

```
        remote_user = models.CharField(default="inactivo", max_length=255)
        status_call = models.IntegerField(default=0)
        canal = models.IntegerField(default=1)
        mensaje_privado = models.CharField(default="null", max_length=255)

        def avatar_url(self):
                return 'http://goflow.alwaysdata.net/images/user16.png'

        class Meta:
                verbose_name = 'User data'
                verbose_name_plural = 'Users data'
......
```

El campo que nos preocupa es el de *mensaje privado*. Los otros campos ya se explicaron en sebcciones anteriores.

4.6. Correción en la visualización del Chat

El problema se puede visualizar en la figura 4.9:

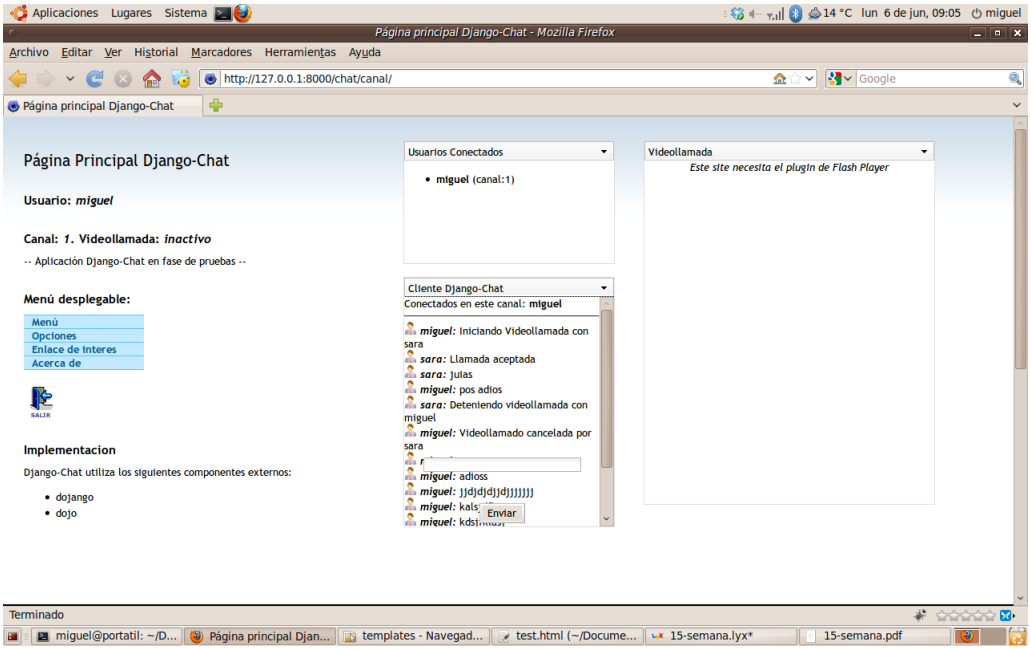

Figura 4.9: Error en la visualización

Como se obseva, la entrada estandar se mezcla con el texto que introduce el usuario. Este error no ha sido solventado por el desarrollador del programa y se ha solucionado sacando la entrada a otra ventana que la situaremos mas abajo. Permitiendo, mediante la barra deslizadora[2] leer los textos que los otros usuarios o él mismo ha introducido, etc. En la figura 4.10se observa el resultado final.

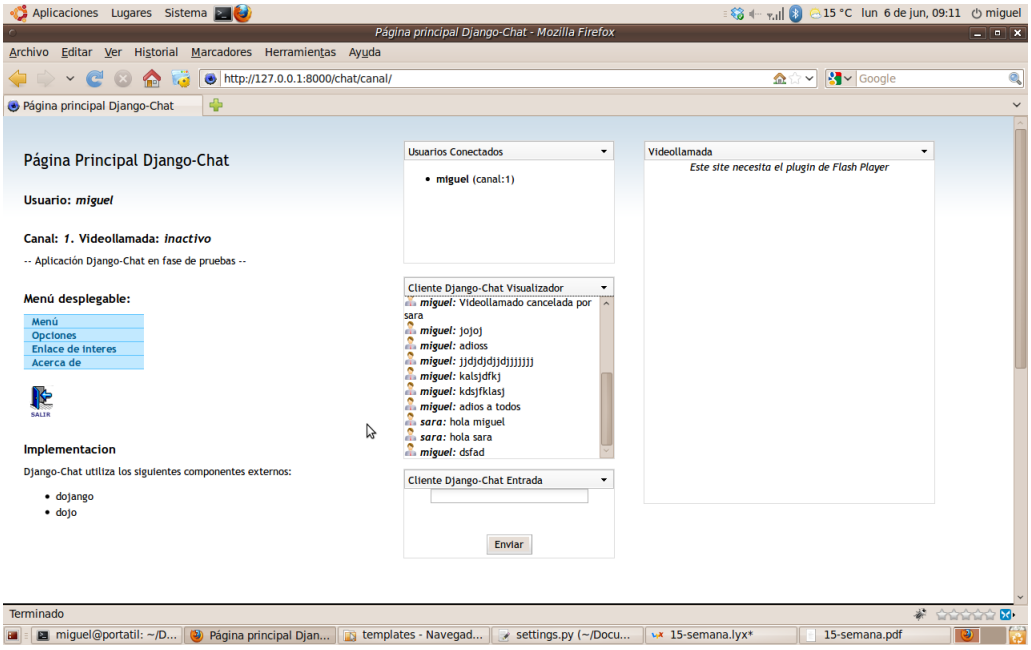

Figura 4.10: Error de visualización solventado

4.7. Correción en la visualización del Chat: barra deslizamiento chat

Solucionando el problema anterior se presentó otro. Un problema bastante molesto y que ha sido solventado, es la barra de deslizamiento en las conversaciones del chat. A continuación vemos en la figura 4.11 la barra de deslizamiento a la que se refiere.

[2]Esta barra deslizadora ha sido también una creación propia, gracias al estudio de javascript.

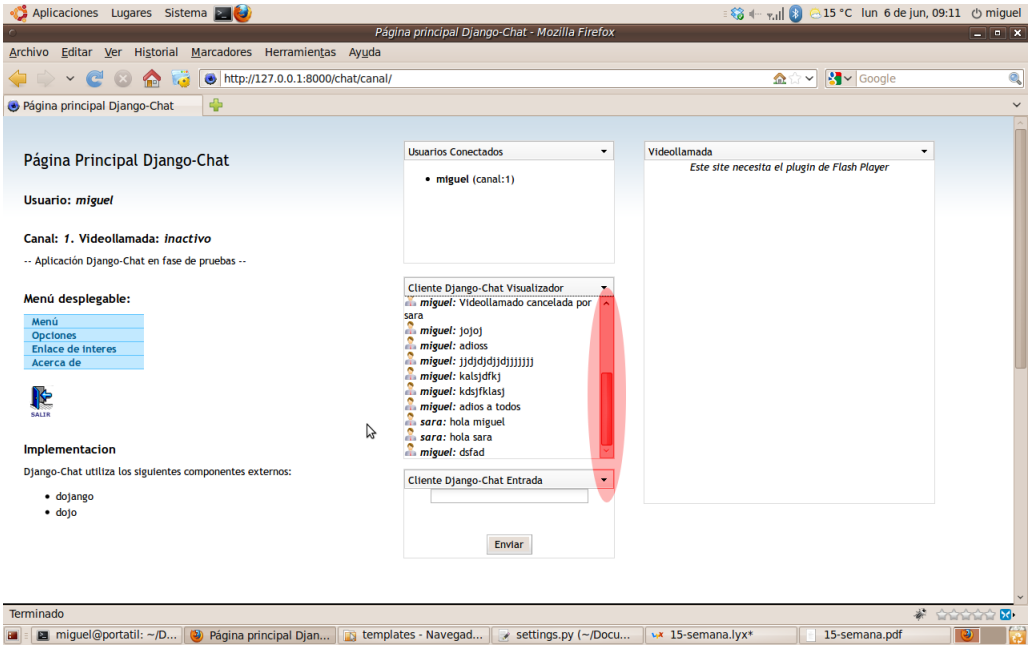

Figura 4.11: Barra deslizamiento

El problema consistía en: un movimiento nulo cuando se actualizaban las conversaciones, es decir, otro usuario, o el mismo, escribía en el canal. La barra deslizadora se quedaba donde estaba sin una visualización de las últimas conversaciones. Para poder verlas había que, manualmente, pinchar con el ratón y desplazar la barra hacia abajo.

La solución que que ha sido tomada ha sido generar un código *JavaScript* que mueve la barra automáticamente hacia abajo. El código añadido al archivo *test.html* es el siguiente:

```
......
<script type="text/javascript">
        function actualizar() {
                var objDiv = document.getElementById("messages");
                objDiv.scrollTop = objDiv.scrollHeight;
        }
</script>
......
```

Desde el propio documento *test.html* se le llama a esta función *actualizar()* que desplaza la barra deslizadora al último mensaje recibido o escrito.

Además se ha modificado el estilo a la capa * * que contiene las conversaciones del canal al que se ha conectado. Se incluye a continuación el código que se emplea para visualizar las conversaciones.

```
......
<!-- Panel Cliente Django-Chat Mensajes-->
<div dojoType="dojox.layout.FloatingPane"  id="chatClient"
        title="Cliente Django-Chat Visualizador" resizable="false"
        dockable="true" closable="false" style="width:260px;
        background:#fff; position:absolute; top:200px; left:500px;">

        <div id="messages" style="height:200px; overflow:auto;">
                Cargando mensajes ...
        </div>

</div>
......
```

En el cuadro anterior se busca un elemento cuyo identificador es *messages*. Podemos ver en este último código *<div id=”messages” ...>*, es decir, el código busca esta capa y modifica sus parámetros para que la barra deslizadora se vaya al final de la capa. ¿Cómo se hace?. Se modifica:

- El estilo de la capa: *style=”....; overflow:auto;”*. Al especificar *auto* este mostrará las barras de scroll segun sean necesarias, horizontales, verticales o las dos, aunque el contenido no lo requiera. Existe más opciones pero esta era la que precisaba para nuestro objetivo.
- Se modifica el parámetro *scrollTop* a scrollHeight, es decir, objDiv.*scrollTop* = *objDiv.scrollHeight*. Este desplaza la barra al final.

Llegar a esta solución, que parece sencilla una vez explicada, ha sido difícil. En internet viene mucha información para solucionar este problema, ninguna trabajaba como se quería. El utilizar el complemento *dojo*, que era una herramienta *JavaScript* que nos ayuda a modificar parámetros de la página *html* fácilmente entre otras cosas, complica algunas operaciones. Al final se ha utilizado una mezcla entre las soluciones que proponían y un poco de “cosecha casera”, ahí está lo bonito de la programación.

4.8. Disminución del uso de la red: disminución de mensajes

4.8.1. Envío de mensajes

Cada vez que el cliente hace un *GET/POST* el servidor contesta actualizando la página del chat solo aquellas partes que interesa. El problema es que el servidor envía los mismos datos cuando nadie escribe nada, provocando una ineficiencia del ancho de banda, este ha sido un cambio al proporcionado por el desarrollador, de forma que, se ha modificado para que no siga enviando los mismos mensajes. El cliente guardará las últimas conversaciones, si no ha habido modificación no se recibirá nada.

Para poder solventar este problema, se ha guardado un bit (*flag_message*) para cada cliente que indica:

0 El servidor no envía información nueva al cliente, el cliente tendrá almacenado las últimas conversaciones.

1 Ha habido modificación, el servidor envía la conversación actualizada.

4.8.2. Optimización

Se recuerda que el servidor registra las conversaciones que los usuarios escriben. Cuando se conecta un cliente le aparece en el visualizador de mensajes del canal, todas las conversaciones que se producen. Un problema que tenía el servidor es que actualiza completamente la pantalla del cliente enviándo nuevamente informacion (conversaciones) que ya disponía el usuario, información que sobrecarga la red "tontamente".

Se ha solventado ya este problema modificando el servidor y el cliente.

Cuando el cliente solicita una actualización de sus parámetros, usuarios conectados, videollamada, mensajes privados, etc, el servidor registra esta solicitud, mirando la fecha desde la última solicitud que hizo este mismo usuario hasta el actual momento. Si hubo cambio en el canal le enviará solo la información que no tiene al cliente. El cliente almacena, realiza un backup, con los mensajes que escribieron otros usuarios o él mismo actualizando éste con las nuevas conversaciones que el servidor le manda. En el cliente siempre habrá información que mostrar. Anteriormente se borraba toda la pantalla y se visualizaba solamente la información que el servidor le mandaba, perdiendo así conversaciones un poco más antigua.

¿Qué código se ha utilizado en el servidor?. Se ha modificado el archivo *views.py* , a groso modo, con las siguientes líneas de código:

```
......
def ordenar(lasts):
        return (lasts.date >= last_req + timedelta(seconds=0.4))

print"\tProcesamiento de mensajes"
lasts = Post.objects.filter(canal =
                int(udata.canal)).order_by('-date').distinct()
print"\tProcesamiento de mensajes terminado"
lasts = filter(ordenar, lasts)
lasts = list(lasts[:30])
......
```

4.9. Hora en los mensajes

Se ha incluido un nuevo campo en la visulización de los mensajes, este nuevo campo es la hora en la que se ha escrito dicho mensaje. En la figura 4.12 podemos ver un ejemplo de este campo:

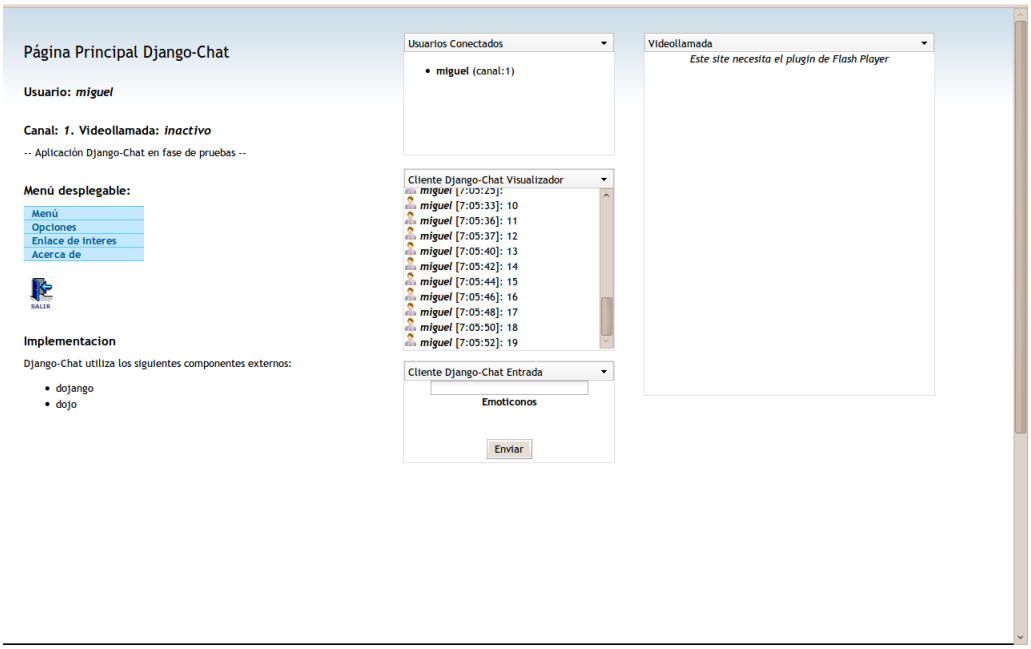

Figura 4.12: Hora en el chat

4.10. Emoticonos

Se ha configurado la aplicación chat para insertar emoticonos en las conversaciones. En la figura 4.13 se aprecia el resultado.

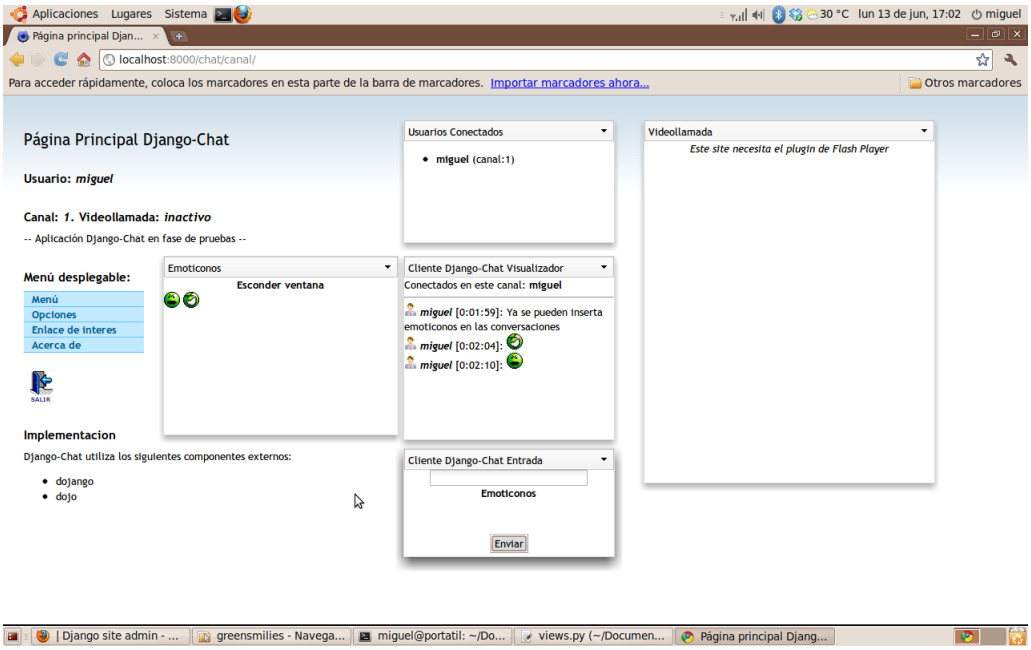

Figura 4.13: emoticonos

Se pueden agregar de dos formas diferentes:

1. Mediante el panel habilitado: haciendo click sobre el emoticono que queremos que aparezca en el mensaje.

2. Mediante teclado: por ejemplo si ponemos :), automáticamente mediante un código *JavaScript* detectará que es un emoticono cambiando los caracteres por la imagen.

Se hace uso de la función *JavaScript*, *buscaremoticono(mensaje):*

```
......
<script type="text/javascript">

  function buscaremoticono(mensaje) {

    mensaje = mensaje.replace(":)",
   "<img src=\"/chat/templates/greensmilies/smiley-yell.gif\" width=\"20\" height=\"20\" border=\"0\">")
    mensaje = mensaje.replace(":D",
   "<img src=\"/chat/templates/greensmilies/smiley-xd.gif\" width=\"20\" height=\"20\" border=\"0\">")

    return(mensaje)
  }

</script>
......
```

4.11. Registro de mensajes del usuario

Se ha investigado la posibilidad de llevar un registro y guardarlo en el ordenador del usuario local, es decir, crearse un archivo con las conversaciones que se quieran guardar.

El lenguaje de programación *JavaScript*, que es el único lenguaje que podemos ejecutar en el navegador web, de forma estandar sin recurrir a otros lenguajes de programación que requieran de una máquina virtual como es *Java,* no posibilita la creación de archivos en el usuario local. No hay nada estandar.

Después de investigar mucho se dió con *ActiveX,* los controles Active X son pequeños bloques empleados para la creación de programas, que se pueden usar para crear aplicaciones distribuidas que funcionen a través de Internet empleando navegadores web. Usando este mecanismo, *ActiveX*, podemos llamar a la función:

```
......
var fso = new ActiveXObject('Scripting.FileSystemObject');
......
```

para luego llamar a las funciones:

fso.OpenTextFile(nombreDelArchivo[,ParaLectura,ParaEscritura,ParaSobreesctura],): Abre un archivo en el modo que pongamos

fso.CreateTexFile(nombreDelArchivo): Crea un archivo

Importante, *ActiveX* solo funciona en navegadores *Internet Explorer*, por lo tanto, hemos modificado la página chat de forma que, se detecte que navegador se utiliza

para que visualice un botón para que pueda llevar la acción de guardar un registro de mensajes. Es decir, en la figura 4.14 se puede ver el botón de *registro de mensajes* que solamente se visualizará si se utiliza *Internet Explorer*.

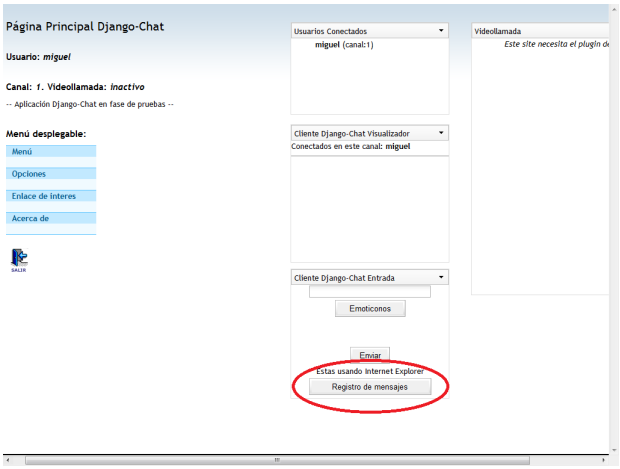

Figura 4.14: Regitro de mensajes

El código javaScript empleado es el siguiente:

```
......
<SCRIPT LANGUAJE="text/javascript">
        function writetofile() {
                var currentTime = new Date()
                var month = currentTime.getMonth() + 1
                var day = currentTime.getDate()
                var year = currentTime.getFullYear()
                var fecha = month + "-" + day + "-" + year

                var situate  = 'd:\\';
                var filename = situate + 'log-message ' + fecha + '.html'
                var fso = new ActiveXObject('Scripting.FileSystemObject');
                if (fso.FileExists(filename)) {
                        var a, ForAppending, file;
                        ForAppending = 8;
                        file = fso.OpenTextFile(filename, ForWriting, false);
                        var aux = dojo.byId("backup").innerHTML
                        file.WriteLine(aux)
                        alert("Sobreescribiendo el registro");
                } else {
                        var file = fso.CreateTextFile(filename, true);
                        var aux = dojo.byId("backup").innerHTML
                        file.WriteLine(aux)
                        alert("Creando un nuevo registro");
                }
                file.Close();
        }
```

```
        </SCRIPT>
......
```

Y el código html agregado al archivo *test.html* para la detección de Internet Explorer y el botón de *registro de mensajes* es el siguiente:

```
......
<!--[if IE]>
        <div style="text-align: center">
                Estas usando Internet Explorer
        </div>

        <div style="text-align: center">
                <button dojoType="dijit.form.Button" onclick="writetofile(); return false;">
                        Registro de mensajes
                </button>
        </div>
<![endif]-->
......
```

Por último, decir que los registros se guardaran con extensión .html, ya que, cuando llamamos a la función *writeline()* le pasamos como parámetro código html, es decir, si no lo guardáramos como html se verían las etiquetas utilizadas en el chat, por ejemplo, no se vería **usuario** en negrita si no que , se vería usuario.

5 Alternativas

5.1. Introducción

En este último capítulo se explica los programas que se han instalado y probados pero que no han sidos usados por un único motivo: se necesitaba alguna forma de comunicar a los usuarios entre sí para utilizar el módulo videollamada, ya que, los programas que se hablan aquí son todos estandarizados y no hay ninguno que hablara sobre videollamada. Una ampliación de este proyecto se deriva en utilizar estos.

Se hablará sobre el protocolo XMPP que es el estandar de mensajería instantanea. BOSH que es una pasarela para poder establecer una conexión a un servidor *XMPP* desde *HTTP*.

Y los programas que permiten utilizar el protocolo XMPP como estandar son:

- En el lado servidor: *Openfire*, *Ejabberd*.
- En el lado cliente mediante javascript: *Stropejs* (se recuerda que no queremos instalar un programa en un sistema operativo y que sirva como cliente de mensajería, si no, utilizar un navegador web).

5.2. Información del protocolo XMPP

XMPP Standards Foundation (XSF) es la fundación que se encarga de la estandarización de las extensiones del protocolo *XMPP*, el estándar abierto de mensajería instantanea con presencia del *IETF*.

5.2.1. Historia

Originalmente la *XSF* se llamaba ***Jabber Software Foundation (JSF).***

5.2.2. ¿Cómo funciona?

XMPP funciona a base de enviar etiquetas *XML*. *XML* es un lenguaje estándar de envío de información a traves de internet. Por ejemplo, HTML (páginas web) es

XML. Se va a explicar brevemente como funciona XML, es muy sencillo:

```
<etiqueta atributo1=valor1>Contenido</etiqueta>
```

donde *"etiqueta"* se denomina *tag* (en inglés) e indica el tipo de elemento que es. El resto de campos son bastantes intuitivos. Un ejemplo real de mensaje *XMPP* sería el siguiente:

```
<message from="miguel@wixet.com" to="otrousuario@wixet.com">
        Esto es un mensaje de prueba de miguel a otrousuario.
</message>
```

Mediante este mensaje *XMPP* se envía un mensaje de *miguel@wixet.com* a *otrousuario@wixet*.com *cuyo* contenido es *"Esto es un mensaje de prueba de miguel a otrousuario."*

5.2.3. Elementos de *XMPP*

En *XMPP* existen tres tipos de elementos diferentes:

***message*:** Son los mensajes que se envían los usuarios entre sí.

***presence*:** Los eventos de presencia de usuarios, cuando se conectan, se desconectan, cuando cambian de estado, de nombre, etc.

***iq*:** Sirve para enviar comandos internos. Por ejemplo para iniciar una nueva conexión, informar de errores, en definitiva de aspectos que no tienen nada que ver con los usuarios.

5.2.4. ¿Qué es *BOSH*?

Las siglas ***BOSH*** viene del inglés ***Bidirectional-streams Over Synchronous HTTP***.

Para iniciar una conexión primeramente se ha de *conectar* a dicho sitio. Para llevar a cabo la *conexión* este proceso entre cliente y servidor se negocian a través del protocolo una sincronización, un cifrado, etc. Este mecanismo es el que utiliza *BOSH.* Es una pasarela para poder establecer una conexión a un servidor *XMPP* desde

HTTP. Esto es necesario por lo siguiente, *XMPP* es una conexión persistente, se mantiene la conexión desde el principio hasta que uno de los dos lados quiera cerrar dicha sesión, entre el inicio y final se envían información. *HTTP* es una conexión de petición y respuesta. Un cliente pide una página web y el servidor la sirve y termina la conexión.

Hay otro método (*polling*) que consiste en enviar muchas peticiones cortas (cada medio segundo por ejemplo). Esta tarea requiere de una sincronización. Esta sincronización se hace mediante un número que se le pasa al servidor en cada petición y que se debe ir incrementando después de cada respuesta. Importante que no sea vacía, más tarde se explicará porqué.

Por ello es por lo que se utiliza *BOSH*. Siempre ha de haber una conexión abierta hacia el servidor por parte del cliente (en este caso un cliente de mensajería). Hay dos formas de conexión con el servidor:

- Enviando una petición (inicio de sesión de un usuario)
- Escuchando en la conexión (se escucha al servidor hasta recibir una respuesta)

El número máximo de conexiones abiertas que se pueden tener, son dos. Si se abre una tercera conexión o un número de sincronización incrementado incorrectamente provoca la finalización de la sesión (el servidor nos desconecta).

Anteriormente se comentó algo sobre *un mensaje vacío*. Se va a llevar una explicación más concisa de la relación que tiene el mensaje vacío con el número de sincronización.

El número de sincronización se llama *RID* y debe ser generado de manera aleatoria. Es importante tener un buen algoritmo que genere un número lo más difícil posible de precedir ya que teniendo el *RID* junto al *SID* (un número que identifica la conexión y se crea en ese momento) alguien podría hacerse pasar por cliente, ya que si se envía un SID y un RID correcto se haría pasar por el cliente (al ser por *HTTP* el servidor no tiene manera de conocer de dónde viene la petición). Se va a describir un ejemplo:

Se inicia sesión por parte del cliente1 y se está a la escucha de un mensaje por parte del servidor (uso de la primera conexión). En cualquier momento se conecta un amigo (cliente2) y se quiere enviar un mensaje de saludo por ejemplo: *"Hola amigo"*. Este genera un objeto *message* que se envía al servidor (uso de la segunda conexión). Si se decide enviar otro mensaje al servidor éste cerraría la sesión. Lo que se debe hacer es cerrar la primera sesión, que devolverá al cliente1 un mensaje vacio. Entonces no se tiene que incrementar *RID*.

El cliente2 al que se le envió *"Hola amigo"* responde diciendo *"Hola como estas"*, el servidor se lo envía al cliente1, creando otro objeto *message* y a través de la conexión creada entre cliente1 y servidor. Como se comentó anteriormente, en *HTTP* una vez recibida la respuesta se finaliza la conexión. En ese momento se debe lanzar una peticion para escuchar al servidor nuevamente, en este caso como se recibe un mensaje no vacio se debe incrementar el RID. Si no se incrementa el servidor lo desconecta.

5.2.5. Posibles implementaciones

- **Como software servidor:** se puede utilizar *Openfire* que es software libre y muy bueno. Se debe configurar. Más adelante se explica la instalación completa en Ubuntu que se llevó a cabo.
- **Como software cliente:** en una página web se indicaba hacer una prueba con el *chat de wixet* este chat cliente se incrusta en cualquier página web. Además se necesita de unos archivos en *Javascript* para el manejo del protocolo *XMPP*:
  - *http://code.google.com/p/wixet/source/browse/branch/wixet3rd/js/jquery.chat-jabber.js*
  - Depende de:
    - Un editor de texto: *http://code.google.com/p/wixet/source/browse/branch/wixet3rd/js/jquery.editor.js*
    - Gestor de ventanas: *http://code.google.com/p/wixet/source/browse/branch/wixet3rd/js/jquery.window.4.03.js*

5.2.5.1. Instalación de Openfire:

Se dedicó bastante tiempo porque en la instalación de la misma se producía un error que no se sabía donde venía. El error, que fue solventado, se buscó en foros y en la página oficial de Openfire. La solución que se adoptó se explica a continuación:

Para llevar a cabo la instalación se ha de seguir los siguiente pasos, por línea de comando en Ubuntu:

```
# Instalamos Apache2 + MySQL5.1 + PHP5 y phpmyadmin
sudo apt-get -y install apache2
sudo apt-get -y install mysql-server mysql-common
sudo apt-get -y install php5 php5-cli
sudo apt-get -y install phpmyadmin
# Para que Apache2 muestre el Error de Host
sudo echo "ServerName localhost" >> /etc/apache2/httpd.conf
# Para que Apache2 Muestre Bien los Asentos y Tildes
sudo echo "AddDefaultCharset ISO-8859-1" >> /etc/apache2/conf.d/charset
```

```
# Reseteamos Apache2
sudo /etc/init.d/apache2 restart
```

Una vez instaladas las dependencias, se procede a instalar Openfire:

```
# Instalamos Java
sudo apt-get install sun-java6-bin
# Configuramos Java como Interprete Principal
sudo update-alternatives --config java
# Creamos el Usuario para OpenFire
sudo adduser openfire
# Descargamos OpenFire en Paquete DEB
wget -c http://download.igniterealtime.org/openfire/openfire_3.7.0_all.deb
# Instalamos OpenFire
sudo dpkg -i openfire_3.7.0_all.deb
# Copiamos Contenido Basico para OpenFire y MySQL
sudo cp /usr/share/openfire/resources/database/openfire_mysql.sql $HOME/
sudo chmod 777 openfire_mysql.sql
# Creamos Base de Datos e Importamos Contenido Básico en MySQL
mysqladmin -h localhost -u root -p create openfire
mysql -h localhost -u root -p openfire < openfire_mysql.sql
# Creamos Usuario y Asignamos Permisos en MySQL
Linea="CREATE USER openfire@localhost IDENTIFIED BY 'CONTRASEÑA';"
echo "$Linea" | mysql -h localhost -u root -p
Linea="GRANT ALL ON openfire.* TO openfire@localhost;"
echo "$Linea" | mysql -h localhost -u root -p
# Eliminamos Archivos Residuales
rm openfire_3.7.0_all.deb
rm openfire_mysql.sql
# Reseteamos OpenFire
sudo /etc/init.d/openfire restart
# Abrimos Administrador Web de OpenFire con firefox http://127.0.0.1:9090
```

5.2.5.2. Información sobre Ejabberd

Otro servidor de mensajería instantanea al estilo de *Openfire* es ***ejabberd***, es de código abierto (*GNU GPL*) para plataformas *Unix* (*BSD*, *GNU/Linux*, etc), *Microsoft Windows* y otras. En las comunicacione se utiliza el protocolo *XMPP*. Está escrito principalmente en *Erlang*, es software concurrente y distribuido.

Alexey Shchepin fundó el proyecto en 2002 y continúa manteniéndolo. El nombre « *ejabberd* » significa « *Erlang Jabber Daemon* ». Se escribe en letras minúsculas solamente, que es uso común en el mundo de Unix. El objetivo del proyecto *ejabberd*

es la creación de un servidor XMPP estable y con variedad de posibilidades.

Instalación de *Ejabberd* Para la instalación de *ejabberd* (v2.0.5-1.1) en *Ubuntu* se utiliza el siguiente comando[1]:

```
# sudo apt-get install ejabberd
```

Configuración básica de *Ejabberd*

Configuración del administrador de *Ejabberd* Para asignar un nombre de administrador necesitamos modificar el archivo */etc/ejabberd/ejabberd.cfg*. Para ello se utiliza el siguiente comando:

```
# sudo nano /etc/ejabberd/ejabberd.cfg
```

En el archivo se añade un usuario administrador *"admin"* en *"localhost"*:

```
.....
%% Admin user
{acl, admin, {user, "admin", "localhost"}}.
```

Salvar con *Ctrl+O* y salir con *Ctrl+X*.

Opciones del comando *ejabberdctl* En el siguiente recuadro se observa todas las opciones de comandos que nos permite *ejabberdctl:*

```
Usage: ejabberdctl [--node nodename] command [options]

Available commands in this ejabberd node:
  status                        get ejabberd status
  stop                          stop ejabberd
  restart                       restart ejabberd
```

[1]Para las siguientes ejecuciones en línea de comandos no se dirá que es Ubuntu

```
reopen-log                      reopen log file
register user server password register a user
unregister user server          unregister a user
backup file                     store a database backup to file
restore file                    restore a database backup from file
install-fallback file           install a database fallback from file
dump file                       dump a database to a text file
load file                       restore a database from a text file
import-file file                import user data from jabberd 1.4 spool file
import-dir dir                  import user data from jabberd 1.4
                                spool directory
delete-expired-messages         delete expired offline messages from database
delete-old-messages n           delete offline messages older than n days
                                from database
mnesia [info]                   show information of Mnesia system
vhost host ...                  execute host-specific commands
srg-list-groups host            list the shared roster groups from host
delete-older-messages days      delete offline messages older than 'days'
vcard-set user host data        set data to content on the vCard
[data2] content
incoming-s2s-number             print number of incoming s2s connections on
                                the node
vcard-get user host data        get data from the vCard of the user
[data2]
status-list status              list the logged users with status
remove-node nodename            remove an ejabberd node from the database
stats onlineusers               number of logged users
stats registeredusers           number of registered users
pushroster-all file             push template roster in file to all those
                                users
srg-get-info group host         get info of a specific group on host
rosteritem-purge [options]      Purge all rosteritems that match filtering
                                options
add-rosteritem user1 server1    Add user2@server2 to user1@server1's roster
user2 server2 nick group subs
srg-delete group host           delete the group
stats uptime-seconds            uptime of ejabberd node in seconds
outgoing-s2s-number             print number of outgoing s2s connections
                                on the node
killsession user server         kill a user session
resource
stats onlineusersnode           number of logged users in the ejabberd node
pushroster file user server     push template roster in file to user@server
load-config file                load config from file
user-resources user server      print user's connected resources
srg-create group host name      create the group with options
description display
get-cookie                      get the Erlang cookie of this node
export2odbc server output
```

```
  directory                      export Mnesia tables on server to files on
                                 output

  connected-users-number         print a number of established sessions
  srg-user-add user server group
  host                           add user@server to group on host
  set-password user server
  password                       set password to user@server
  delete-older-users days        delete users that have not logged in the last
                                 'days'
  rem-rosteritem user1 server1
  user2 server2                  Remove user2@server2 from
                                 user1@server1's roster
  compile file                   recompile and reload file
  status-num status              number of logged users with status
  push-alltoall server group     adds all the users to all the users in Group
  srg-user-del user server
  group host                     delete user@server from group on host
  connected-users                list all established sessions

Examples:
  ejabberdctl restart
  ejabberdctl --node ejabberd@host restart
  ejabberdctl vhost jabber.example.org ...
```

Opción destacada de *ejabberdctl*: Registrar usuarios *Ejabberd* viene con un comando de administrador *ejabberdctl*, con el que se puede configurar nuevos usuarios. Para crear un *"usuario"* en *"localhost"* con el password *"secret"*, se ejecuta el siguiente comando:

```
# sudo ejabberdctl register usuario localhost secret
```

y para asignar un password "secret" al usuario *"admi*n" sería:

```
# sudo ejabberdctl register admin localhost secret
```

Opción destacada de *ejabberdctl:* Eliminar usuarios Para poder eliminar un "usuario" se ejecuta el siguiente comando:

```
# sudo ejabberdctl unregister usuario localhost
```

Administrador Web de *Ejabberd* Para administrar ejabberd y monitorizar los usuarios conectados, permisos, etc podemos irnos al servidor web alojado en:

http://localhost/xmpp-httpbind

Tecleamos:

- usuario: admin@localhost
- password: secret
    - Nota: Un fallo muy común y que llevó bastante tiempo en detectar fue no poner *@localhost*

En la figura 5.1 podemos ver una vista previa del *administrador web ejabberd:*

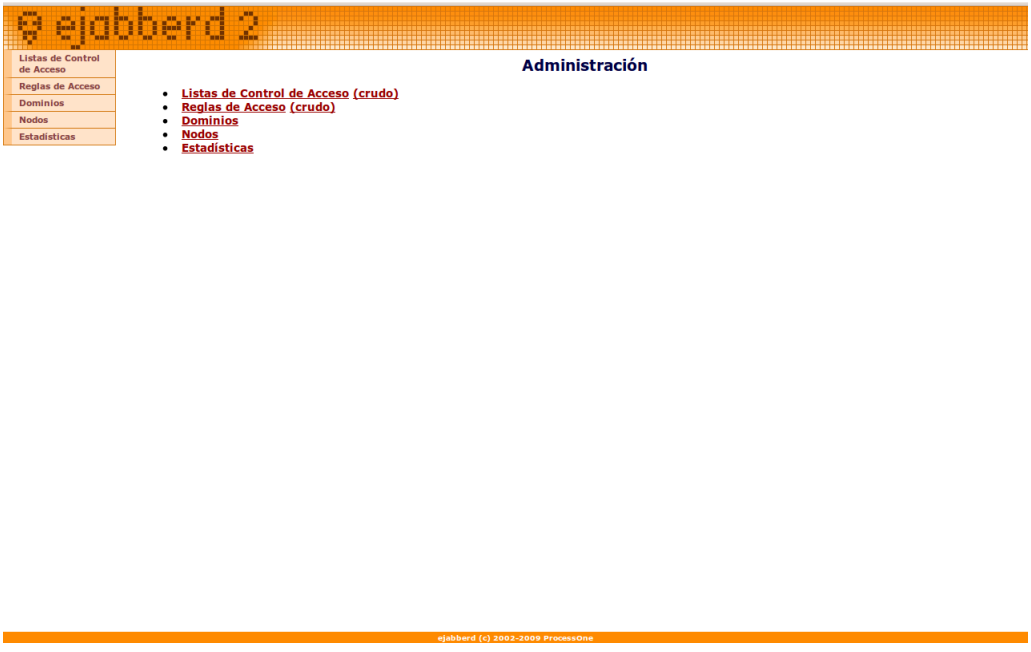

Figura 5.1: Administrador Web de *ejabberd*

Configurar *ejabberd* para el soporte del protocolo *XMPP* Una vez funcionando el servidor en la máquina *"localhost"* y se halla configurado un par de usuarios, se necesita identificarlos cuando soliciten una petición de conexión, es decir autenticarlos y para ello se utiliza el protocolo *BOSH* para la comunicación con el servidor *XMPP* con *http-bind*. Es decir, *BOSH* actua de pasarela del mundo HTTP al mundo XMPP. Se edita el archivo */etc/ejabberd/ejabberd.cfg*, con el siguiente comando:

```
# sudo nano /etc/ejabberd/ejabberd.cfg
```

El archivo debería quedar con estas líneas:

```
.....
{5280, ejabberd_http, [
        http_bind,
        http_poll,
        web_admin
]}
.....
```

Hay que añadir el módulo *http_bind* para soportar esa pasarela:

```
.....
{modules, [
.....
        {
                mod_http_bind, []},
.....
]}
.....
```

Y por último reiniciamos el servidor:

```
# sudo ejabberdctl restart
```

Verificación del módulo *http_bind:* Para verificar la instalación se escribe en una página web http://localhost:5280/http-bind.

Se debería ver la figura 5.2:

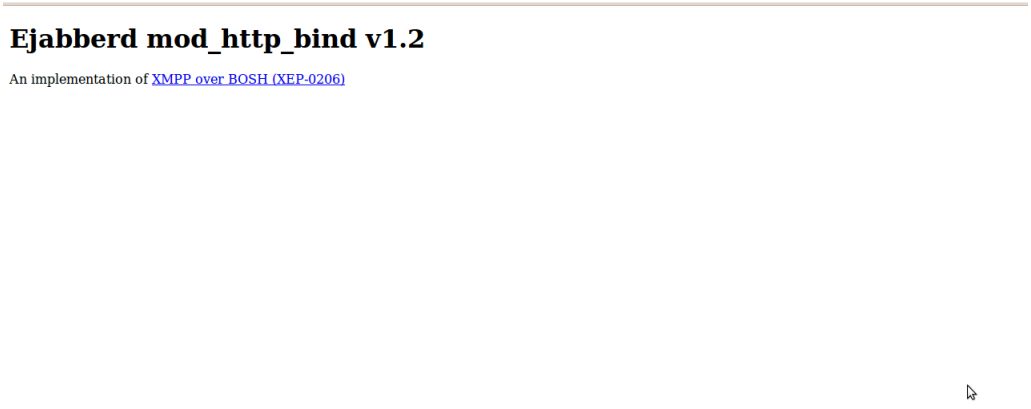

Figura 5.2: Verificación del módulo *http_bind*

Instalación de Nginx A continuación se procede a la instalación del servidor web *nginx*. *nginx* es un servidor web como es *Apache* pero con la particularidad de que se obtuvo más informacion acerca de *nginx* que de *Apache* en cuanto a la interacción *nginx-ejabberd* (servidor web-servidor mensajeria). La idea principal es que, independientemente de cual se utilice, la función que tendrá es redirigir (hacer proxy) la petición de conexión del cliente *XMPP* al servidor correspondiente, *Openfire* o *Ejabberd*. Se procede a instalar el servidor web *Nginx*, se ejecuta el siguiente comando para ello:

```
# sudo apt-get install nginx
```

Por defecto *nginx* utiliza el puerto 80 como escucha. Se cambia al puerto 8181 por comodidad. Esto se hace principalmente por si se utiliza otro servidor web como es *Apache* el cual se tenga instalado y no se quiera cambiar su configuración. Por defecto la raiz del directorio donde *nginx* alojará su páginas web y las servirá está en */var/www/nginx-default*. Para realizar los cambios que se ha comentado se ha de modificar el archivo */etc/nginx/sites-available/default*, con el siguiente comando:

```
# sudo nano /etc/nginx/sites-available/default
```

El archivo deberá quedar como el siguiente:

```
.....
server {
        listen 8181 default;
        server_name localhost;
.....
location / {
        root /var/www/nginx-default;
        index index.html index.htm;
}
.....
```

Se reinicia *nginx*:

```
# sudo nginx -s reload
```

Redirección xmpp-httpbind en *nginx* Modificación del archivo */etc/nginx/sites-available/default* para redireccionar la solicitud de conexión del cliente *XMPP* al servidor *ejabberd*.

```
# sudo nano /etc/nginx/sites-available/default
```

El archivo deberá quedar como el siguiente:

```
.....
server {
        location /xmpp-httpbind {
                proxy_pass http://localhost:5280/http-bind;
        }
.....
}
.....
```

Se reinicia *nginx*:

```
# sudo nginx -s reload
```

5.2.5.3. Instalación de Strophejs

Una vez instalado *ejabberd* (servidor xmpp) y *nginx* (servidor web que redirecciona la solicitud xmpp del cliente al servidor *ejabberd*), se procede a la "instalación"/descarga del programa cliente xmpp desde la página web oficial:

http://code.stanziq.com/strophe/

o mediante comando ejecutar:

```
# git clone git://code.stanziq.com/strophejs
```

se descarga, pero tiene que estar accesible desde el servidor web *nginx*, más tarde se comentará donde hay que colocar dichos archivos si se usara *apache* como servidor web. Se crea un enlace simbólico en la carpeta raíz del servidor web *nginx:*

```
# sudo ln -s    /var/www/nginx-default/strophejs-1.0.1
                /ruta_completa_a_strophejs-1.0.1/
```

Strophe es una familia de librerías escrita para clientes *XMPP*. *strophe.js* es una librería pura en JavaScript. El escenario es el siguiente:

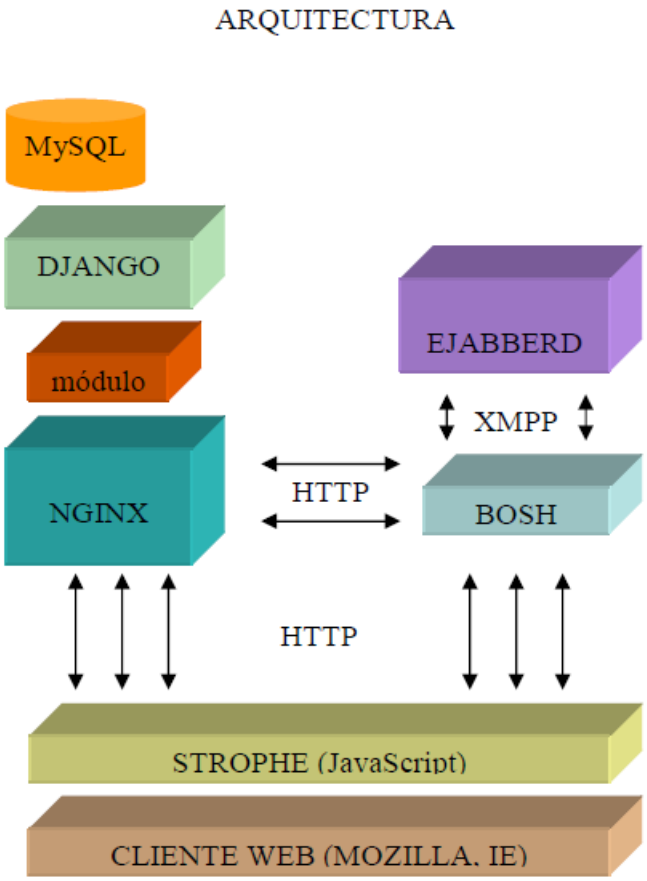

Figura 5.3: Arquitectura ejabberd-nginx

En la figura 5.3 se mencionan las siglas *BOSH*. Ya comentado en la subsección 5.2.4. A modo de resumen, es un protocolo de transporte que emula la semántica de una conexión TCP bidireccional entre dos entidades usando eficientemente mútiples parejas request/response. Puede ser que se entienda mejor en inglés, la definición que nos dan es:

"is a transport protocol that emulates a bidirectional stream between two entities (such as a client and a server) by using multiple synchronous HTTP request/response pairs without requiring the use of polling or asynchronous chunking."

Ejemplos

Uso del ejemplo que viene en la carpeta */strophe-1.0.1/examples/echobot.html*
Esto es un ejemplo de uso de *strophejs* que conecta con el servidor ejabberd. Para ello primeramente en ejabberd crearemos un usuario por ejemplo *"miguel"* el dominio será *"localhost"* y contraseña la que queramos. A continuación abriremos el enlace:

http://localhost:8181/strophejs/examples/echobot.html

y el resultado final es la página web tal como la figura 5.4:

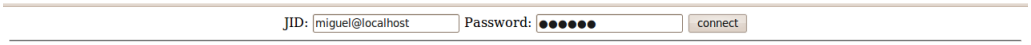

Figura 5.4: Conexión con ejabberd con echobot.html

JID: ***miguel@localhost***

Password: ******

y se clickea en connect. Aparece el siguiente mensaje:

Strophe is connecting.

Strophe is connected.

ECHOBOT: Send a message to miguel@localhost/256495807413019400612397 12 to talk to me.

Observaciones: puerto 8181 como comentamos en apartados anteriores y la ruta donde se encuentra strophe (carpeta que se ha descargado).

Empathy Empathy es un cliente de mensajería gratuito que viene en ubuntu que es capaz de conectarse con servidores ejabberd utilizando el protocolo xmpp. Al instalar el programa nos pregunta si queremos crear una cuenta nueva o conectarnos con una cuenta ya creada. Elegimos esta última opción y elegimos del menú desplegable jabber como servidor de mensajerica (otros son msn, irc, google talk, etc).

Usuario: miguel@localhost (si utilizamos ejabberd como servidor xmpp, podemos utilizar también openfire como servidor)

Password: el que le hayamos dado.

5.2.6. Misma arquitectura con tupla Openfire - Apache. Configuración

Una vez instalado *openfire* se configura *apache* para que sirva las peticiones de los clientes que utilizan el protocolo *xmpp* para que los redirija a openfire. La configuración es la siguiente:

5.2.6.1. Activación del módulo *proxy.load*

Con el siguiente comando se activa dicho módulo, se situa en */etc/apache2/*mods-available/:

```
# sudo a2enmod proxy_http
```

5.2.6.2. Configuración del archivo */etc/apache2/mods-available/proxy.conf*

Se añaden las siguientes líneas:

```
<IfModule mod_proxy.c>
        #turning ProxyRequests on and allowing proxying from all may allow
        #spammers to use your proxy to send email.

        ProxyRequests Off
        <Proxy *>
                AddDefaultCharset off
                Order deny,allow
                Allow from all  // línea editada
                #Allow from .example.com
        </Proxy>
```

```
        // Estados dos líneas hay que introducirlas
        ProxyPass /http-bind http://127.0.0.1:7070/http-bind/
        ProxyPassReverse /http-bind http://127.0.0.1:7070/http-bind/

        # Enable/disable the handling of HTTP/1.1 "Via:" headers.
        # ("Full" adds the server version; "Block"
                # removes all outgoing Via: headers)
        # Set to one of: Off | On | Full | Block
        ProxyVia On
</IfModule>
```

5.2.6.3. Configuración del archivo */strophe-1.0.1/examples/echobot.js*

Cuando se vaya a utilizar este ejemplo se tiene que editar la primera línea por la siguiente:

```
var BOSH_SERVICE = '/http-bind';
```

ya que, *Openfire* al contrario de ejabberd no solicita la url:
http://localhost/xmpp-httpbind
sino esta:
http://localhost/http-bind
por último, al igual que había que introducir los usuarios en *ejabberd*, en *openfire* se tiene que agregar los usuarios que se van a conectar.

NOTA: En la configuración de Openfire los usuarios pueden ser usuarios dado: por un archivo externo con la utilización del protocolo *LDAP.*

5.3. Programas instalados

Esta es una recopilación de los programas instalados en el sistema operativo linux Ubuntu que por una o varias razones han dado su aporte:

- Python versión 2.6
- Epydoc: para la elaboración de documentación automática

- BlueFish Editor 1.0.7: Bluefish es un editor para programadores y diseñadores web, además es un proyecto open source con licencia GPL.
- IDLE(usando Python v2.6): Entorno de Desarrollo Integrado para Python
- Paquetes necesarios para la distribución de paquetes Python (en Win, RPM y codigo fuente)
- Instalación de *setuptools* para la distribución de aplicaciones Python en formato *egg*, archivos mediante los que distribuir aplicaciones en Python
- Django
- Instalación de la Base de Datos MySQL con el Administrador
- Elaboración de la documentación con LyX

Índice de figuras

Índice de cuadros

Bibliografía

[1] HOLOVATY, ADRIAN - KAPLAN MOS, JACOB (2010): *"La guía definitiva de django"*. Editorial Anaya Multimedia Apress.

[2] DUBOIS, PAUL (2008): *"Edición Especial MySQL"*

[3] SUMMERFIELD, MARK (2009): *"Python3"*. Editiorial Anaya Multimedia.

[4] Python Programming Language - Oficial Website
http://www.python.org/

[5] Django chat - Djongo
http://code.google.com/p/django-chat/

[6] Python Programming Language - Oficial WebsiteChat server & client - Example
http://code.activestate.com/recipes/531824-chat-server-client-using-selectselect/

[7] Python Programming Language - Oficial WebsiteTutorial de Python en español
http://mundogeek.net/tutorial-python/

[8] Python Programming Language - Oficial WebsiteTutorial de JavaScript en español
http://www.librosweb.es/javascript/pdf/

[9] CD Suplementario. Carpeta *Libros*: *libro-javascript.pdf*

[10] CD Suplementario. Carpeta *Libros*: *libro-python.pdf*

[11] CD Suplementario. Carpeta *Libros*: *libro-xhtml.pdf*

[12] CD Suplementario. Carpeta *Libros*: *libro-ajax.pdf*

[13] CD Suplementario. Carpeta *Libros*: *libro-django.pdf*

A Anexo: Análisis de la comunicaciones con Wireshark

A.1. Acceso a la aplicación chat: identificación de usuario y de canal

Conversación01: Descarga de la página web */chat/*. En el fichero conversacion01.txt ya se observa un error, y es la peticion de la página web cuya ruta es /chat/. El servidor no encuentra nada en dicha ruta y no devuelve nada.
CD://capturas-wireshark/conversacion/conversacion01.txt

Conversación02: Redirección a la ruta url */chat/accounts/login/?next=/chat/,* descarga de la página para logueo
CD://capturas-wireshark/conversacion/conversacion02.txt

Conversación03: Descarga del archivo de presentación y estilos */media/css/login.css*
CD://capturas-wireshark/conversacion/conversacion03.txt

Conversación04: Requiere del archivo de imagen *favicon.ico*, pero no se encuentra, este fallo no es problema de la instalación, si no que, al descargarse el código fuente, este archivo no existe.
CD://capturas-wireshark/conversacion/conversacion04.txt

A.2. Página principal del chat: uso del chat

Conversación05: Aquí nos da un sessionID
CD://capturas-wireshark/conversacion/conversacion05.txt

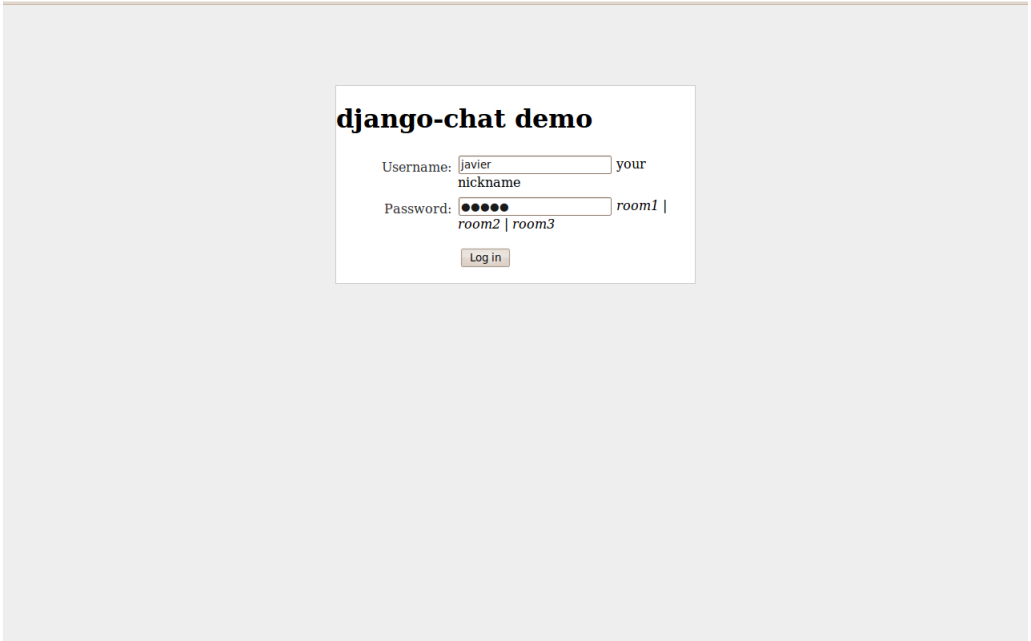

Figura A.1: Acceso a la aplicación chat

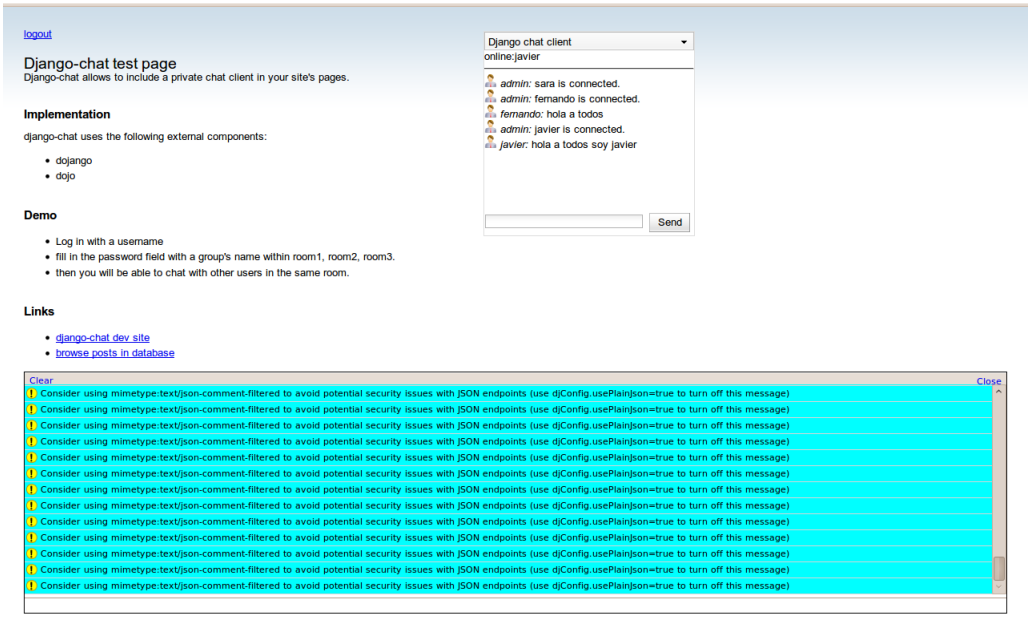

Figura A.2: Página principal del chat

Conversación06: Descargamos la página chat con el sessionID
CD://capturas-wireshark/conversacion/conversacion06.txt

Conversación07: Aquí observamos un fallo, no se descarga el archivo *dojango.js* ya que la página en la conversación anterior nos descargamos la página html que tenía una etiqueta script, necesita este archivo para que funcione correctamente.
CD://capturas-wireshark/conversacion/conversacion07.txt

Conversación08: En este segundo intento por descargarse el archivo *dojango.js* si que lo consigue.
CD://capturas-wireshark/conversacion/conversacion08.txt

Conversación09: Requiere del archivo de imagen *favicon.ico*, pero no se encuentra, este fallo no es problema de la instalación, si no que, al descargarse el código fuente, este archivo no existe.
CD://capturas-wireshark/conversacion/conversacion09.txt

Conversación10: Accede a la página */chat/update/* , es decir, para ver si ha habido cambios en la página, es decir, si otro usuario ha escrito algo, una nueva conexión por parte de otros usuarios, etc.
CD://capturas-wireshark/conversacion/conversacion10.txt

Conversación11: Accede nuevamente a la página */chat/update/.* Cada cierto tiempo se actualiza para realizar lo comentado anteriormente.
CD://capturas-wireshark/conversacion/conversacion11.txt

Conversación12: accede nuevamente a la página */chat/update/*
CD://capturas-wireshark/conversacion/conversacion12.txt

Conversación13: Idem al anterior, accede nuevamente a la página */chat/update/*
CD://capturas-wireshark/conversacion/conversacion13.txt

Conversación14: Idem al anterior, accede nuevamente a la página */chat/update/*
CD://capturas-wireshark/conversacion/conversacion14.txt

Conversación15: Idem al anterior, accede nuevamente a la página */chat/update/*
CD://capturas-wireshark/conversacion/conversacion15.txt

Conversación16: Envío de un mensaje de un usuario a otros usuarios. Este mensaje se guardará permanentemente en el servidor. Cuando un usuario accede de nuevo, este mensaje aparecerá.
CD://capturas-wireshark/conversacion/conversacion16.txt

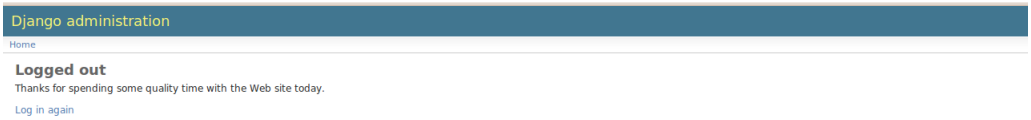

Figura A.3: Desconexión de la aplicación chat

Conversación17: Accede nuevamente a la página */chat/update/*
CD://capturas-wireshark/conversacion/conversacion17.txt

Conversación18: Accede nuevamente a la página */chat/update/*
CD://capturas-wireshark/conversacion/conversacion18.txt

Conversación19: Accede nuevamente a la página */chat/update/*
CD://capturas-wireshark/conversacion/conversacion19.txt

Conversación20: Accede nuevamente a la página */chat/update/*
CD://capturas-wireshark/conversacion/conversacion20.txt

A.3. Finalización de la aplicación

Conversación21: Se descarga la página */chat/admin/logout/* para finalizar la aplicación
CD://capturas-wireshark/conversacion/conversacion21.txt

Conversación22: Se descarga el archivo de presentación y hojas de estilos */media/css/base.css*
CD://capturas-wireshark/conversacion/conversacion22.txt

Conversación23: Se descarga el archivo */media/img/admin/nav-bg-reverse.gif*
CD://capturas-wireshark/conversacion/conversacion23.txt