

2018

Proyecto de Mensajería Segura: BirBeep



Autores:

- Rubén Enrique García Benito
- Sebastián Andrés Faranna
- Luis Miguel Sánchez-Clemente Batres

Profesor coordinador:

- Miguel Ángel Muñoz Herrera

8-6-2018

Índice

1. Resumen.....	1
2. Datos de la aplicación	
a) Datos técnicos.....	2
b) Funcionalidad.....	3
i. Generación certificados.....	4
ii. Protocolo SSL.....	6
iii. Creación del sobre digital.....	7
iv. Apertura del sobre digital.....	9
v. Servidor.....	10
vi. Cliente.....	12
vii. Peticiones.....	13
viii. Modelo E/R.....	19
ix. Diagrama Relacional.....	20
c) Usabilidad.....	21
3. Conclusiones.....	23
4. Recursos.....	23
5. Bibliografía.....	24

1. Resumen

La información a día de hoy es uno de los activos más importantes que las empresas necesitan proteger. Por otra parte, para resolver los diferentes proyectos, es cada vez más habitual disponer de equipos multidisciplinares dispersos geográficamente, que obviamente precisan de mecanismos que les posibiliten una comunicación fluida. Por todo ello, un sistema de mensajería que comunique al personal de la organización de manera confidencial, garantizando al mismo tiempo la autenticación de emisores y receptores de la información, aportaría sin duda un valioso servicio a la empresa; es lo que vamos a desarrollar en este proyecto. Se utilizará una arquitectura cliente-servidor para resolver el sistema:

- El servidor actuará de intermediario en la distribución de cada mensaje.
- Los clientes serán los remitentes o destinatarios de cada comunicación, y se utilizarán sobres digitales para asegurar la autenticación de emisor, la confidencialidad, la integridad y el no repudio.

2. Datos de la aplicación

a) Datos técnicos

- **Nombre de la aplicación**

El nombre de la aplicación es Birbeep.

- **Plataforma**

El lenguaje elegido es Java, ya que es el que más hemos utilizado en clase.

La plataforma que utilizamos es Windows. Para usar la aplicación debemos tener instalado JRE(Java Runtime Enviroment) para que puedan correr los programas creados con lenguaje Java.

El servidor corre bajo el JRE, que es un entorno en tiempo de ejecución de Java creado por la Máquina Virtual de Java (JVM). La Máquina Virtual de Java es una máquina virtual de proceso nativo, es decir, ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial (el bytecode Java), el cual es generado por el compilador del lenguaje Java. El JRE actúa como mediador entre el sistema operativo y Java.

Las aplicaciones que han sido utilizadas para desarrollar la aplicación son Eclipse, para el desarrollo del código de aplicación junto con el servidor e interfaz gráfica de usuario, y phpMyAdmin, para gestionar la base de datos.

Eclipse es una plataforma de software compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para



desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha

sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado *Java Development Toolkit* (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse). Le hemos utilizado mayormente para la creación del código de la aplicación y del servidor.

PhpMyAdmin es una herramienta escrita en PHP con la intención de manejar la administración de MySQL a través de páginas web, utilizando Internet. Actualmente puede crear y eliminar Bases de Datos, crear, eliminar y alterar tablas, borrar, editar y añadir campos, ejecutar cualquier sentencia SQL, administrar claves en campos, administrar privilegios, exportar datos en varios formatos y está disponible en 72 idiomas.



- **Requisitos mínimos del sistema**

Estos serían los requisitos mínimos para lanzar esta aplicación:

Windows:

- Windows 7 o posteriores
- Privilegios de administrador para la instalación
- Internet Explorer 9 y posteriores, Firefox 3.6 y posteriores

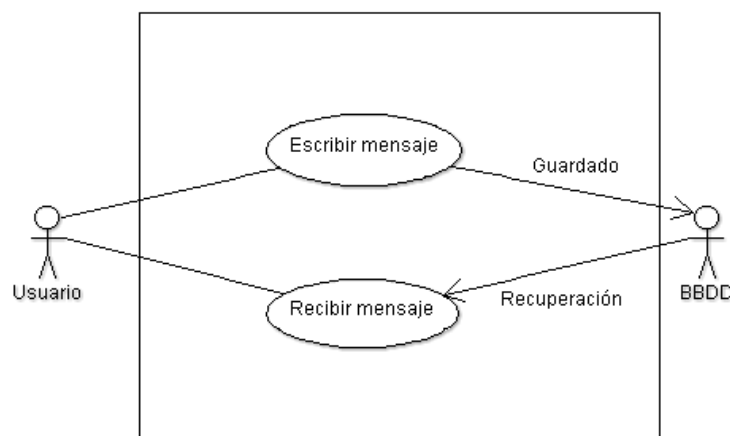
Mac OS X:

- Mac OS X 10.7.3+, 10.8.3+, 10.9+
- Privilegios de administrador para la instalación
- Internet Explorer 9 y posteriores, Firefox 3.6 y posteriores

b) Funcionalidad

Las funcionalidades son:

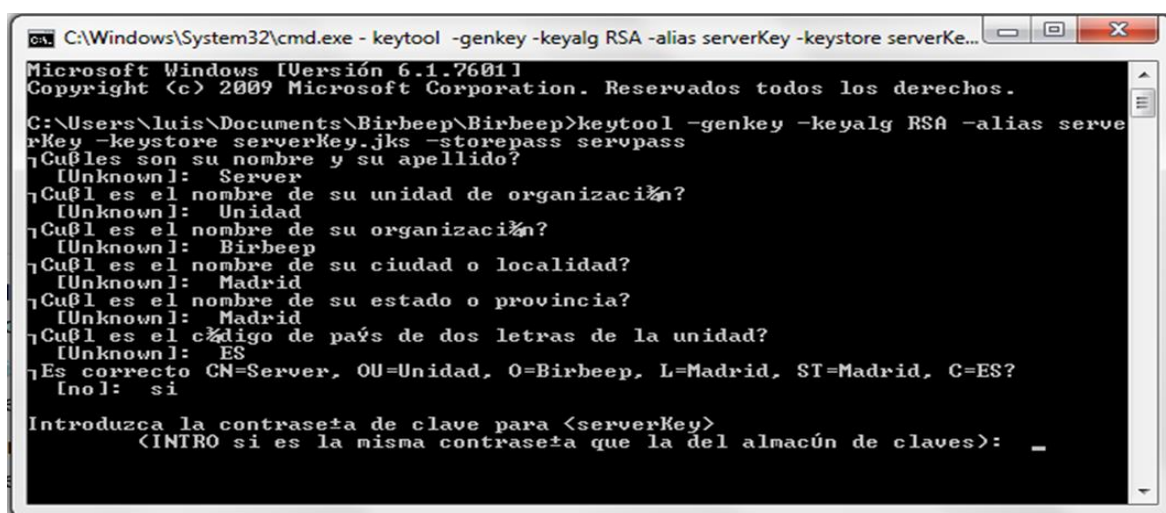
- Básicamente lo que permite esta aplicación es el intercambio de mensajes entre usuarios dados de alta previamente en el sistema, la peculiaridad del programa ante el que nos encontramos es que asegura la confidencialidad de las comunicaciones, cifrando la conexión entre el cliente y servidor además de encriptar toda la información que es transferida por los flujos y guardada en la base de datos.



- Se hace uso de certificados digitales, que es un fichero informático con una estructura de datos que contienen información sobre una entidad, firmado por otra entidad, considerada por una colección de otras entidades como una autoridad para este tipo de contenido.

Se generan mediante la herramienta **keytool**. Tiene diferentes pasos:

1. Creamos el certificado del Servidor con su almacén de certificados correspondiente.



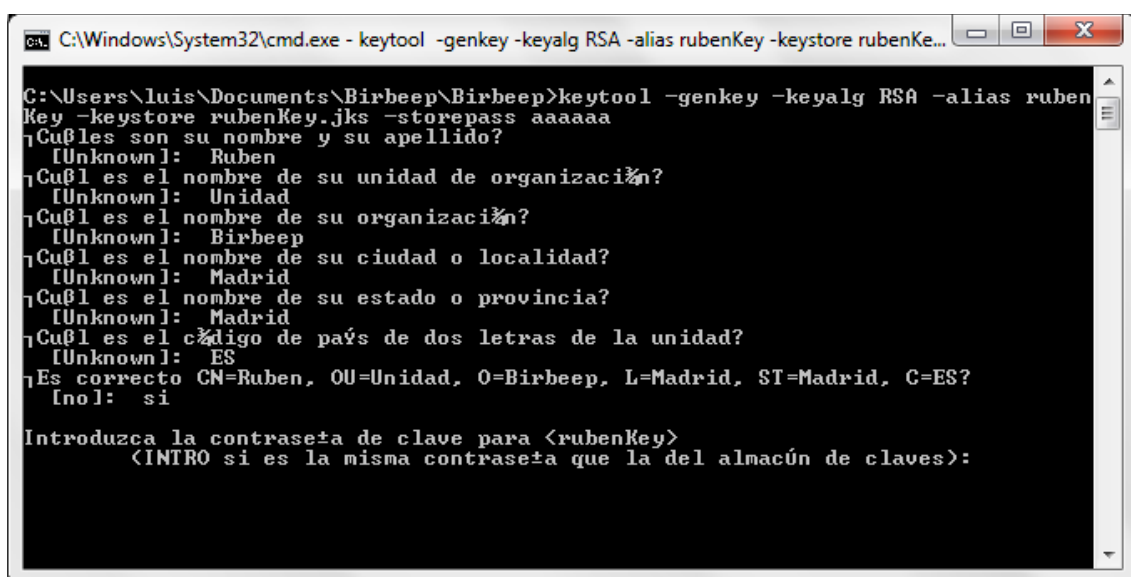
```
C:\Windows\System32\cmd.exe - keytool -genkey -keyalg RSA -alias serverKey -keystore serverKey.jks
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.

C:\Users\luis\Documents\Birbeep\Birbeep>keytool -genkey -keyalg RSA -alias serverKey -keystore serverKey.jks -storepass serupass
¿Cuáles son su nombre y su apellido?
[Unknown]: Server
¿Cuál es el nombre de su unidad de organización?
[Unknown]: Unidad
¿Cuál es el nombre de su organización?
[Unknown]: Birbeep
¿Cuál es el nombre de su ciudad o localidad?
[Unknown]: Madrid
¿Cuál es el nombre de su estado o provincia?
[Unknown]: Madrid
¿Cuál es el código de país de dos letras de la unidad?
[Unknown]: ES
¿Es correcto CN=Server, OU=Unidad, O=Birbeep, L=Madrid, ST=Madrid, C=ES?
[no]: si

Introduzca la contraseña de clave para <serverKey>
<INTRO si es la misma contraseña que la del almacén de claves>: _
```

Posteriormente, este almacén de Certificados albergará los certificados del total de los usuarios que están registrados en la aplicación.

2. Creamos el certificado del primer cliente con su almacén correspondiente.

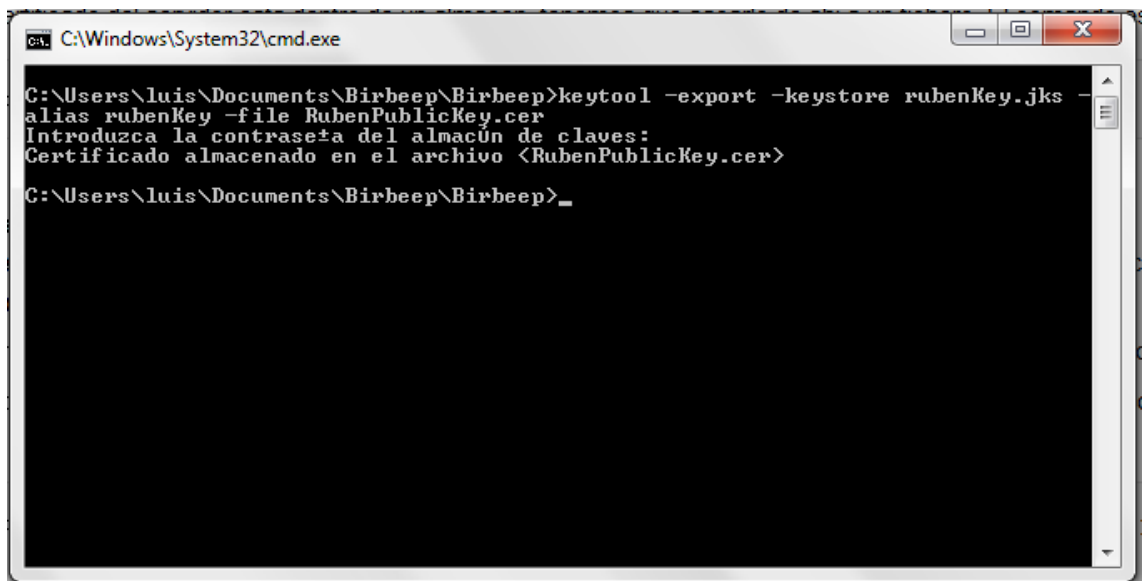


```
C:\Windows\System32\cmd.exe - keytool -genkey -keyalg RSA -alias rubenKey -keystore rubenKey.jks
C:\Users\luis\Documents\Birbeep\Birbeep>keytool -genkey -keyalg RSA -alias rubenKey -keystore rubenKey.jks -storepass aaaaaa
¿Cuáles son su nombre y su apellido?
[Unknown]: Ruben
¿Cuál es el nombre de su unidad de organización?
[Unknown]: Unidad
¿Cuál es el nombre de su organización?
[Unknown]: Birbeep
¿Cuál es el nombre de su ciudad o localidad?
[Unknown]: Madrid
¿Cuál es el nombre de su estado o provincia?
[Unknown]: Madrid
¿Cuál es el código de país de dos letras de la unidad?
[Unknown]: ES
¿Es correcto CN=Ruben, OU=Unidad, O=Birbeep, L=Madrid, ST=Madrid, C=ES?
[no]: si

Introduzca la contraseña de clave para <rubenKey>
<INTRO si es la misma contraseña que la del almacén de claves>:
```

Esta tarea se llevará a cabo en la máquina del usuario que está registrado en la aplicación, y más adelante se exportará el certificado del almacén al almacén de certificados de **confianza** del servidor.

3. Exportamos el certificado del cliente

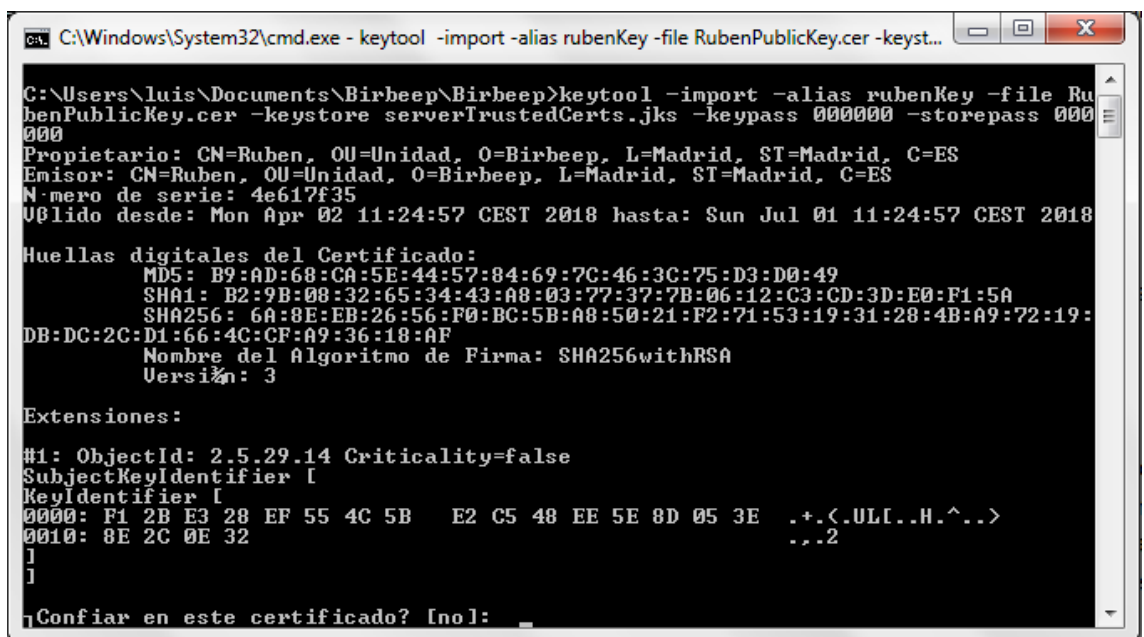


```
C:\Windows\System32\cmd.exe

C:\Users\luis\Documents\Birbeep\Birbeep>keytool -export -keystore rubenKey.jks -
alias rubenKey -file RubenPublicKey.cer
Introduzca la contraseña del almacén de claves:
Certificado almacenado en el archivo <RubenPublicKey.cer>

C:\Users\luis\Documents\Birbeep\Birbeep>_
```

4. Y lo copiamos en el almacén de certificados de confianza del Servidor.



```
C:\Windows\System32\cmd.exe - keytool -import -alias rubenKey -file RubenPublicKey.cer -keyst...

C:\Users\luis\Documents\Birbeep\Birbeep>keytool -import -alias rubenKey -file Ru
benPublicKey.cer -keystore serverTrustedCerts.jks -keypass 000000 -storepass 000
000
Propietario: CN=Ruben, OU=Unidad, O=Birbeep, L=Madrid, ST=Madrid, C=ES
Emisor: CN=Ruben, OU=Unidad, O=Birbeep, L=Madrid, ST=Madrid, C=ES
Número de serie: 4e617f35
Válido desde: Mon Apr 02 11:24:57 CEST 2018 hasta: Sun Jul 01 11:24:57 CEST 2018

Huellas digitales del Certificado:
MD5: B9:AD:68:CA:5E:44:57:84:69:7C:46:3C:75:D3:D0:49
SHA1: B2:9B:08:32:65:34:43:08:03:77:37:7B:06:12:C3:CD:3D:E0:F1:5A
SHA256: 6A:8E:EB:26:56:F0:BC:5B:A8:50:21:F2:71:53:19:31:28:4B:A9:72:19:
DB:DC:2C:D1:66:4C:CF:A9:36:18:AF
Nombre del Algoritmo de Firma: SHA256withRSA
Versión: 3

Extensiones:
#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: F1 2B E3 28 EF 55 4C 5B E2 C5 48 EE 5E 8D 05 3E .+<(.UL[.H.^..>
0010: 8E 2C 0E 32 ...2
]
]

¿Confiar en este certificado? [no]: _
```

5. Se repiten los pasos 2 y 3 para cada cliente
6. Exportamos, en esta ocasión, el certificado del Servidor con la orden del paso número 3.
7. Como en el paso 4, ahora CADA cliente tendrá su almacén de confianza en el que estará el certificado Servidor.
De esta manera, el cliente guarda en su almacén el certificado del Servidor.

- Utilización del protocolo SSL, que es un protocolo criptográfico, que proporciona comunicaciones por una red.

Consiste en el uso de criptología asimétrica para autenticar a la contraparte con quien se están comunicando y para intercambiar una llave simétrica. Esta sesión es luego usada para cifrar el flujo de datos entre los dos partes. Esto permite la confidencialidad del mensaje, códigos de autenticación de mensajes para integridad y como producto lateral, autenticación del mensaje.

Lo utilizamos para la comunicación entre cliente-servidor

El uso de Sockets SSL en lugar de los corrientes difiere en que es necesaria la creación de un contexto:

```
SSLContextsc = SSLContext.getInstance("TLS");
```

Posteriormente lo inicializamos y le indicamos de dónde sacará los certificados. Para terminar la creación del Socket SSL hay que crear una factoría (patrón de diseño que define una clase constructora Abstracta, con unos métodos abstractos y otros no) sobre la cual obtendremos el Socket de dialogo (serverSocket) o en el caso del cliente el socket que realiza la conexión(client):

```
sc.init(keyManagers, trustManagers, null);

SSLSocketFactory ssf = sc.getSocketFactory();

//Cliente
SSLSocket client = (SSLSocket) ssf.createSocket(address,
port);
//Servidor
ServerSocket serverSocket = ssf.createServerSocket(port);
```

"keyManagers" y *"trustManagers"* son los almacenes.

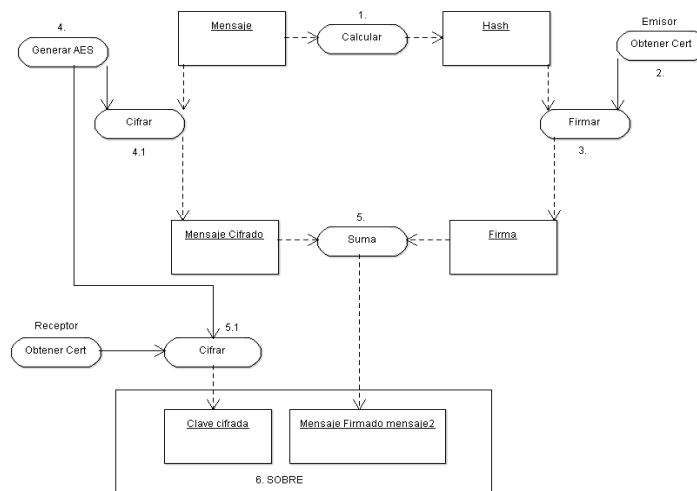
Para terminar se realiza el intercambio de certificados:

```
client.startHandshake();
```

Está operación nos devuelve un SSLSocket, pero como es una clase hija de Socket podemos utilizar esté objeto que retorna como un Socket corriente.

- Utilizamos los sobres digitales, que se basa en que su cifrado y su descifrado utilizan la clave pública y la clave privada. Lo usamos para el envío de mensajes y en la seguridad en la aplicación.

Cifrar



Las fases para cifrar son:

1. Se **calcula el hash** con el mensaje (String pasado como parámetro a la clase que engloba esta lógica). El hash es una función que cifra una entrada y actúa de forma parecida a las funciones hash, ya que comprime la entrada a una salida de menor longitud y es fácil de calcular. Se caracteriza por cumplir propiedades que las hacen idóneas para su uso en sistemas que confían en la criptografía para dotarse de seguridad. Estas propiedades las hacen resistentes frente ataques maliciosos que intentan romper esa seguridad.

```

MessageDigest digest = MessageDigest.getInstance("SHA-256");
byte[] hash = digest.digest(mensaje);
  
```

2. Se **obtiene la clave privada** del emisor, que se encuentra en el almacén del certificado.

Para ello se obtiene una instancia de la clase keystore pasándole el tipo de almacén:

```

Keystore ks = KeyStore.getInstance("JKS");
  
```

Posteriormente se obtiene a través del flujo de entrada de un fichero el propio archivo de certificado. Se carga mediante el método load del keystore el archivo obtenido por el flujo de entrada pasándole la clave del certificado:

```

ks.load(FileInputStream, "Clave certificado");
  
```

3. A continuación, **se firma el hash** con la clave privada, para autenticar que ese mensaje está escrito por el emisor y que no ha sido modificado en la trayectoria de envío.

De esto se encarga la clase Cipher, la cual necesita instanciarse pasándole un string con el tipo de cifrado ("RSA/ECB/PKCS1Padding") para luego proceder al encriptado de la instancia de Cipher pasándole la clave privada obtenida en el paso anterior. Un método para terminar nos proporcionará una "firma" en un array de bytes:

```
Cipher ciph = Cipher.getInstance("RSA/ECB/PKCS1Padding");
ciph.init(Cipher.ENCRYPT_MODE, privateKey);
byte[] firma = ciph.doFinal(hash);
```

4. Se genera la clave AES para hacer el **cifrado simétrico** y se cifra el mensaje con esa clave. La clave AES es un esquema de cifrado por bloques adoptado como un estándar de cifrado por el gobierno de los Estados Unidos.

```
KeyGenerator keyGenerator =
KeyGenerator.getInstance("AES");
keyGenerator.init(128);
Key key = keyGenerator.generateKey();
Cipher aes = Cipher.getInstance("AES/ECB/PKCS5Padding");
aes.init(Cipher.ENCRYPT_MODE, key);
byte[] mensaje_cifrado = aes.doFinal(mensaje);
```

5. Obtenemos la **suma del mensaje cifrado con su hash firmado**. Después cogemos la clave pública del receptor y ciframos la clave generada.

```
byte[] mensaje2 = new byte[mensaje_cifrado.length + 256];
System.arraycopy(mensaje_cifrado, 0, mensaje2, 0,
mensaje_cifrado.length);
System.arraycopy(firma, 0, mensaje2,
mensaje_cifrado.length, 256);
Cipher ciph = Cipher.getInstance("RSA/ECB/PKCS1Padding");
ciph.init(Cipher.ENCRYPT_MODE, publicKey);
byte[] encriptado2 = ciph.doFinal(key.getEncoded());
```

6. Obtenemos el sobre firmado

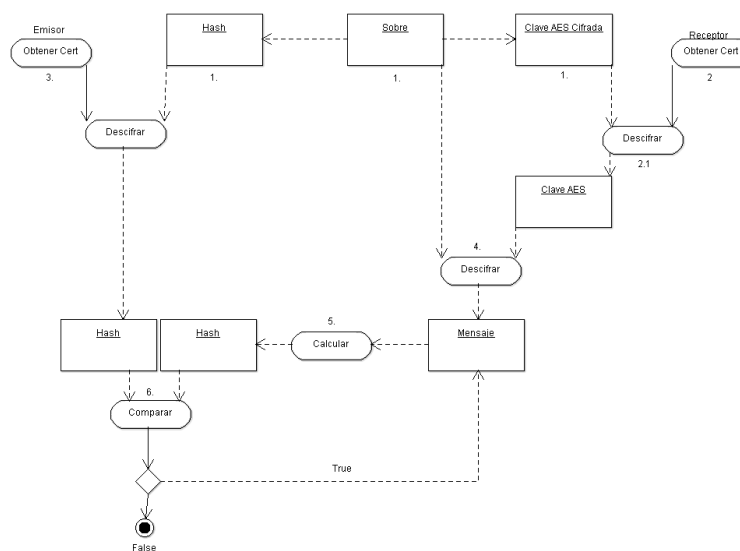
Para esto, se añade a la suma del mensaje cifrado con el hash en el paso anterior una nueva cantidad de bytes:

```
byte[] mensaje3 = new byte[mensaje2.length + 256];
```

Se clonan los arrays generados pasándoles respectivamente: el mensaje previo, la posición desde donde empezará la copia de este mensaje previo, el destino, la posición desde donde se empezará la adición de los nuevos bytes, el número de elementos copiados.

```
System.arraycopy(mensaje2, 0, mensaje3, 0,
mensaje2.length);
System.arraycopy(encriptado2, 0, sobre, mensaje2.length,
256);
```

Descifrar



Las fases de descifrado son:

1. **Obtenemos el sobre** digital enviado por el emisor del mensaje. Este sobre contiene el mensaje encriptado, el hash firmado y la clave encriptada.

```
byte[] mensaje = newbyte[sobre.length - 512];
```

Aquí se hace el proceso invertido del último apartado del cifrado, es decir, según la posición desde donde nos coloquemos en el array vamos a recuperar la información precisa.

```
byte[] firma = newbyte[256];
byte[] encodedKey = newbyte[256];
System.arraycopy(sobre, 0, mensaje, 0, sobre.length - 512);
System.arraycopy(sobre, sobre.length - 512, firma,
0, 256);
System.arraycopy(sobre, sobre.length - 256, encodedKey,
0, 256);
```

2. **Obtenemos la clave privada del receptor**(como en el paso 2 del cifrado) y **desciframos la clave simétrica** (como en el paso 3 del cifrado pero pasando a Cipher "DECRYPT_MODE").

```
byte[] bkey = cipher.doFinal(encodedKey);  
Key key = newSecretKeySpec(bkey, 0, 16, "AES");
```

3. Se **obtiene la clave pública del emisor**. Ésta se ha recibido mediante una petición al servidor.
4. **Desciframos el mensaje**. (Igual que el paso 4 del cifrado pero pasando a Cipher "DECRYPT_MODE").
5. **Calculamos el hash del mensaje**. (Igual que el paso 1 del cifrado)
6. **Comparamos** el hash generado con el que viene del sobre digital para saber si ha sido modificado el mensaje.

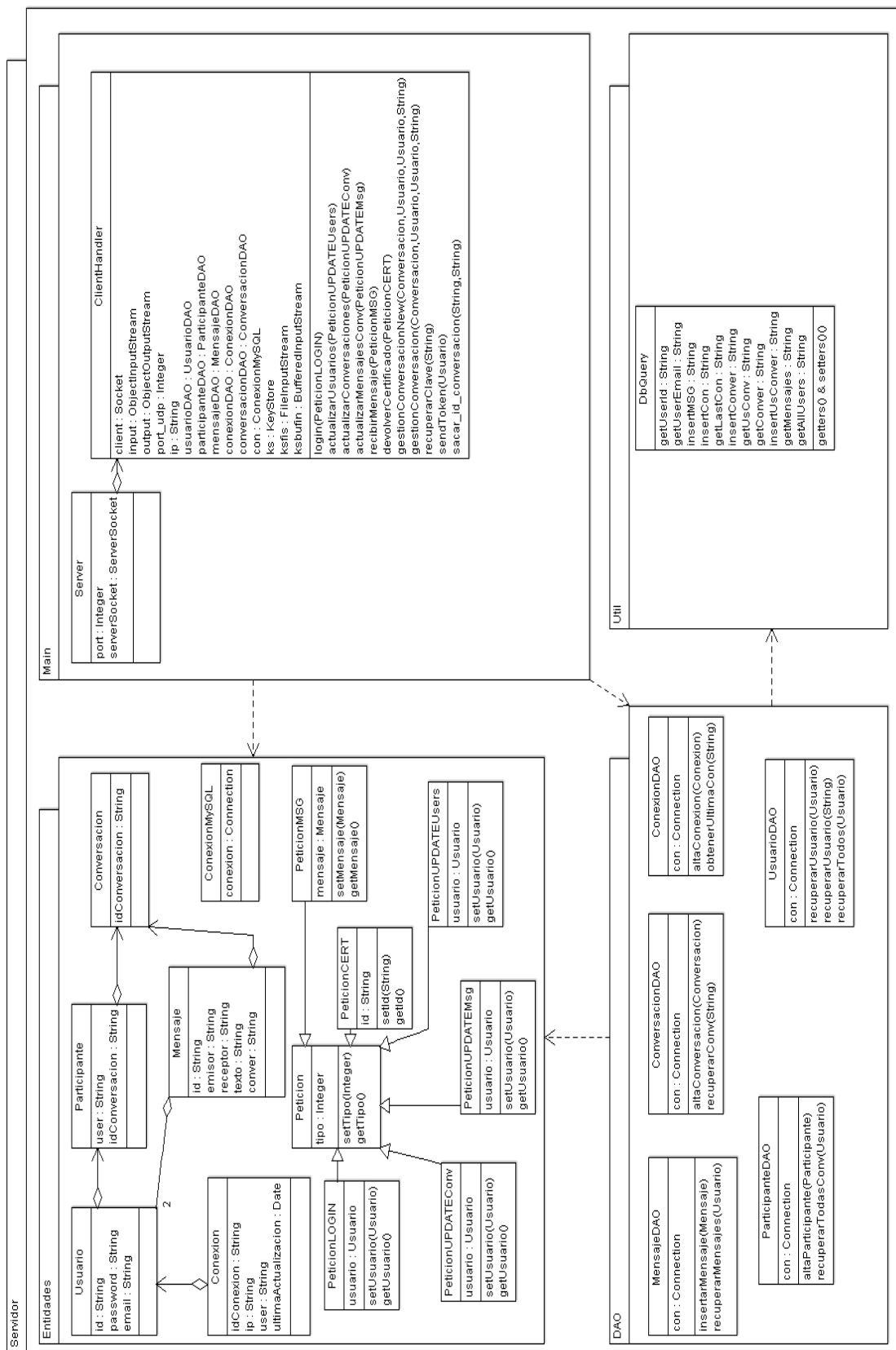
- El **servidor** corre en la máquina virtual de Java. Cada vez que se inicia la aplicación y se entra en la aplicación, se crea una conexión SSL con el servidor. Si no se consigue conectar con el servidor se envía un mensaje de error.

Está conectado a una Base de Datos de tipo MySQL y todos los mensajes serán guardados en esta Base de Datos totalmente cifrados para garantizar la confidencialidad.

Para atacar a la capa de persistencia se ha utilizado el patrón de diseño DAO (Data Access Object, por sus siglas en Inglés), de esta manera se aísla la capa de negocio de la de persistencia facilitando así las posibles modificaciones en la lógica del programa.

En la imagen a continuación podemos ver la estructura de clases de la aplicación, con sus dependencias y asociaciones.

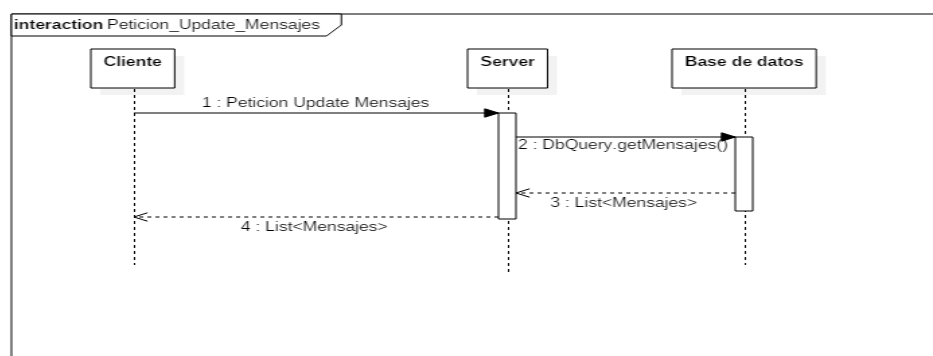
Se ha implementado el uso de hilos para el tratamiento de los Socket, de esta manera el Servidor se encuentra en un bucle escuchando las diferentes llamadas por parte de los clientes.



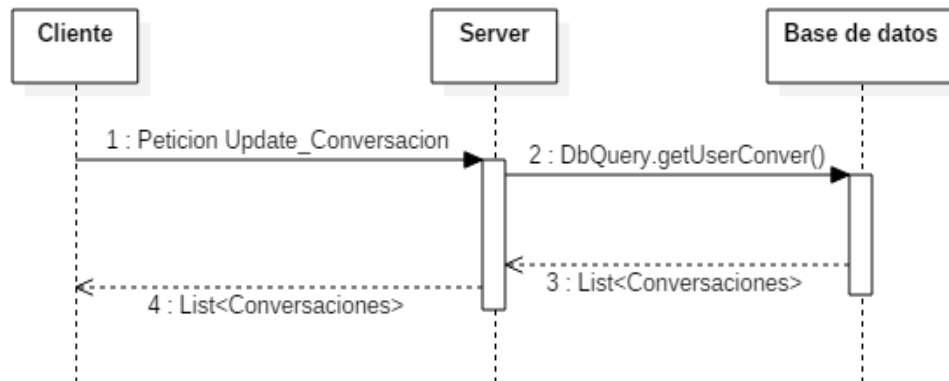
- El **cliente** de la misma manera que el servidor, entra en un bucle mediante un hilo creado en el main de la clase llamadora que es suspendido hasta que se produce una nueva acción por parte del usuario despertando el hilo que se encarga de tramitar las peticiones. La interfaz gráfica inicializa la clase principal del lado cliente, esta clase principal (SSLConexion) crea un nuevo hilo que este sí, es el encargado de tramitar las peticiones, que vienen dadas por las interacciones del usuario con la interfaz gráfica y algunas otras (Actualización de conversaciones y usuarios) integradas en la propia lógica de la aplicación. El primer paso cuando se arranca el hilo creado (clase Sender.java que extiende de Thread) es ponerlo a dormir, de esta manera, hasta que en la interfaz, provocado por un evento del usuario, no se interrumpe el hilo dormido, no comienza el intercambio de datos entre el servidor y el lado cliente.

Cuando se interrumpe el hilo que está durmiendo lo siguiente a realizar es enviar al servidor que tipo de información espera recibir el cliente, este envío de info se realiza serializando las clases "Petición" diferenciando los tipos.

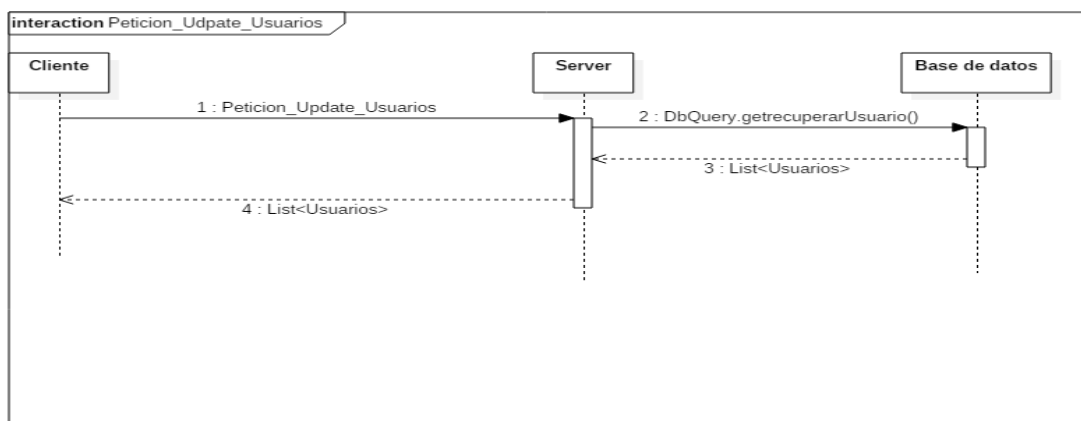
- Puede hacer diferentes peticiones al servidor, que son actualizarse, actualizar mensajes, actualizar conversaciones, actualizar usuarios, petición mensaje, pedir un certificado, login y logout. Para el uso de estas peticiones tenemos que deserializar los datos que nos devuelven, para lo cual utilizamos flexjson-2.1.
 - La petición **actualizarse** es llamada cuando el servidor envía un token UDP, que avisa que hay nuevos mensajes para el usuario, y hace que se actualice. Esta puede ser que llame a otra petición que actualice los usuarios, los mensajes o una conversación.
 - La petición **actualizar mensaje** recupera todos los mensajes de una conversación.



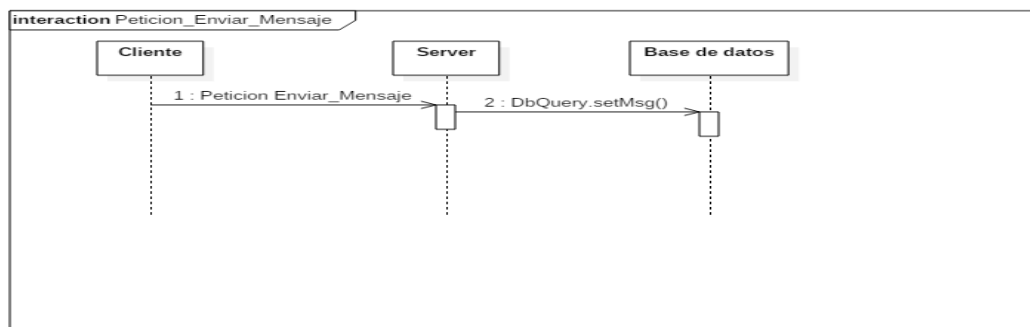
- o La petición **actualizar conversaciones** recupera todas las conversaciones en las que el usuario ha sido participante.



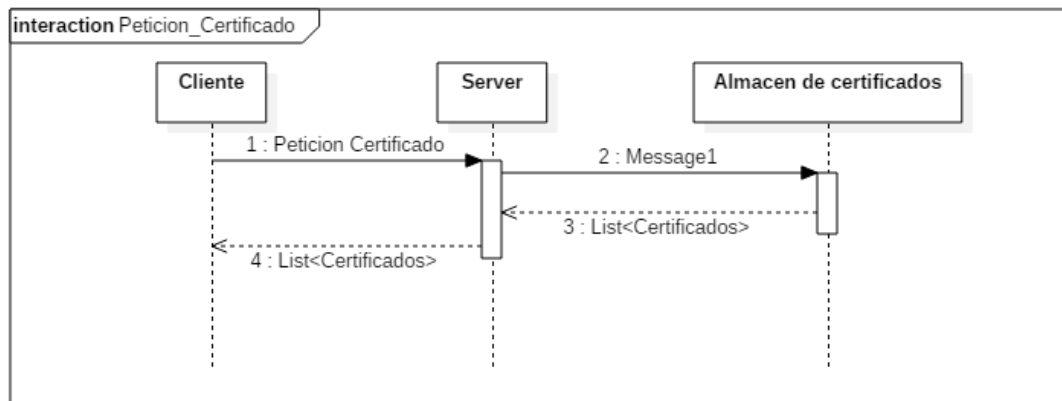
- o La petición **actualizar usuarios** recupera todos los usuarios que están dados de alta en la aplicación, devolviendo una lista de usuarios sacada de la base de datos.



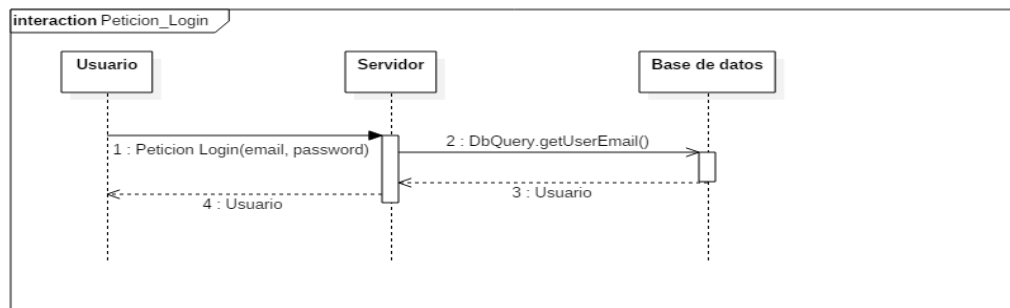
- o La petición **enviar mensaje** es la que hace el envío al receptor y al mismo emisor y el guardado del mensaje en la base de datos. El envío del mensaje al mismo emisor es para saber qué mensajes envía.



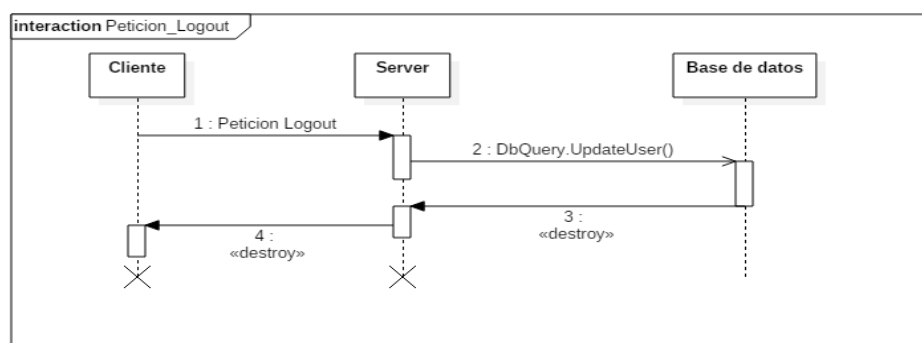
- o La petición **pedir un certificado** trae una lista de todos los certificados de los usuarios pertenecientes a la conversación excepto el del usuario, que lo tiene instalado en su ordenador propio. Esta petición es llamada cuando se envía un mensaje y se necesita la clave pública del receptor del mensaje, que está en el certificado.

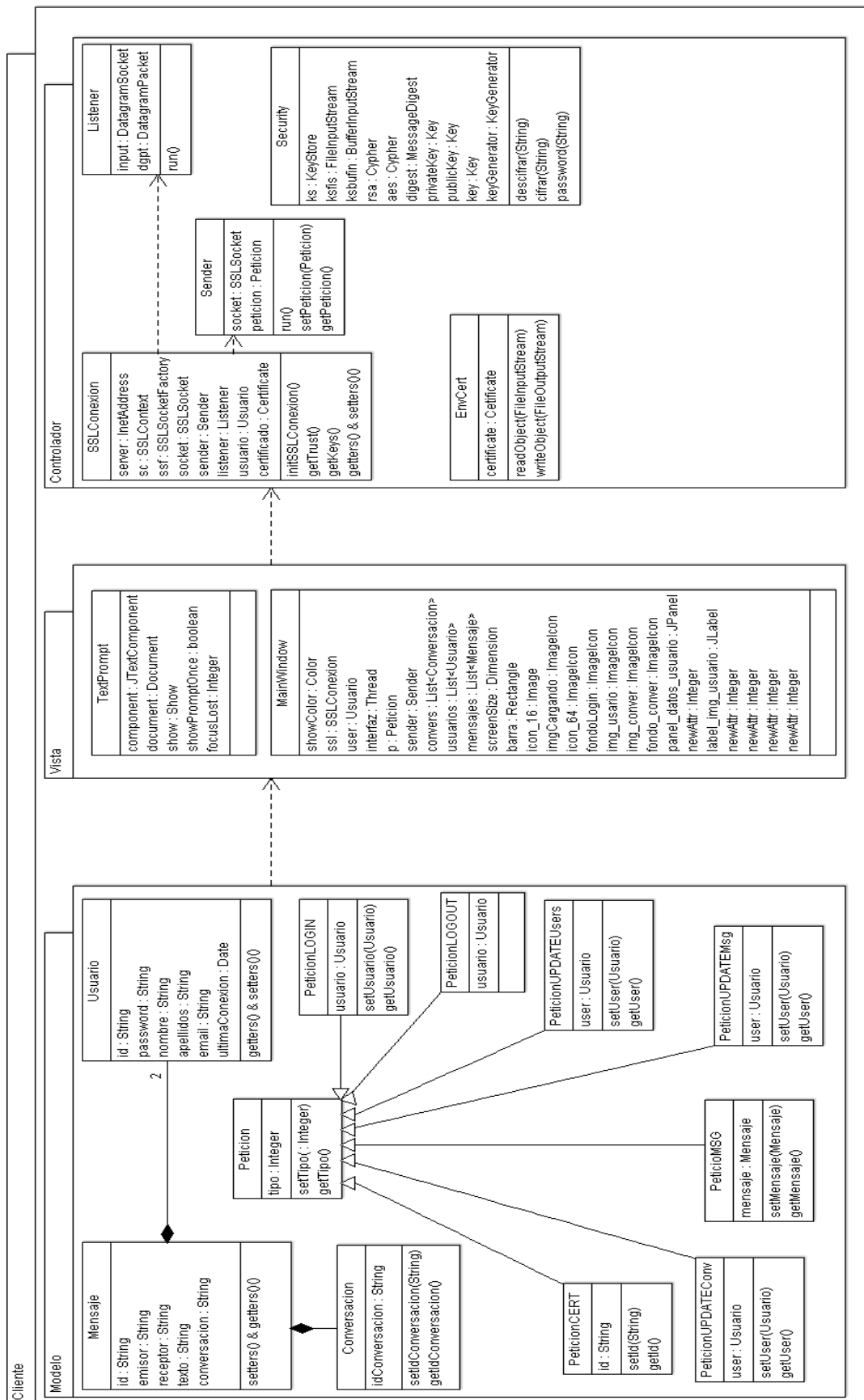


- o La petición **login** se usa cuando inicia la aplicación porque es la que comprueba si existe ese usuario con esa contraseña en la base de datos. Si ese usuario está en la base de datos, el servidor crea un hilo para que se encargue de ese usuario hasta este que cierre la aplicación.



- o La petición **logout** se usa cuando se cierra la sesión del usuario que esté utilizando la aplicación. Cierra el hilo que nos asigna el servidor, pero el servidor sigue corriendo.





- Hay un flujo de información entre el cliente y el servidor. Este flujo está basado en peticiones, que han sido explicadas en el punto previo, y en la serialización. La serialización consiste en un proceso de codificación de un objeto en un medio con el fin de transmitirlo a través de una conexión en red como una serie de bytes o en un formato humanamente más legible, que en nuestro caso es JSON. Cada petición devuelve un objeto, que hay que deserializar para poder usarlo. Las peticiones, que son las clases con los datos para ser serializadas, cuentan con una clase Padre de la que heredan el resto de tipos de Petición. Cuando se envía una petición desde el cliente, ésta se ha definido con un valor entero que informa de su tipo, además de otros valores necesarios para el intercambio de información. Cuando llega al servidor a través de los Sockets seguros, está petición concreta se deserializa en una genérica, la clase padre:

```
Peticion petition = new
JSONDeserializer<Peticion>().deserialize(obj.toString(),Peticion.class
);
```

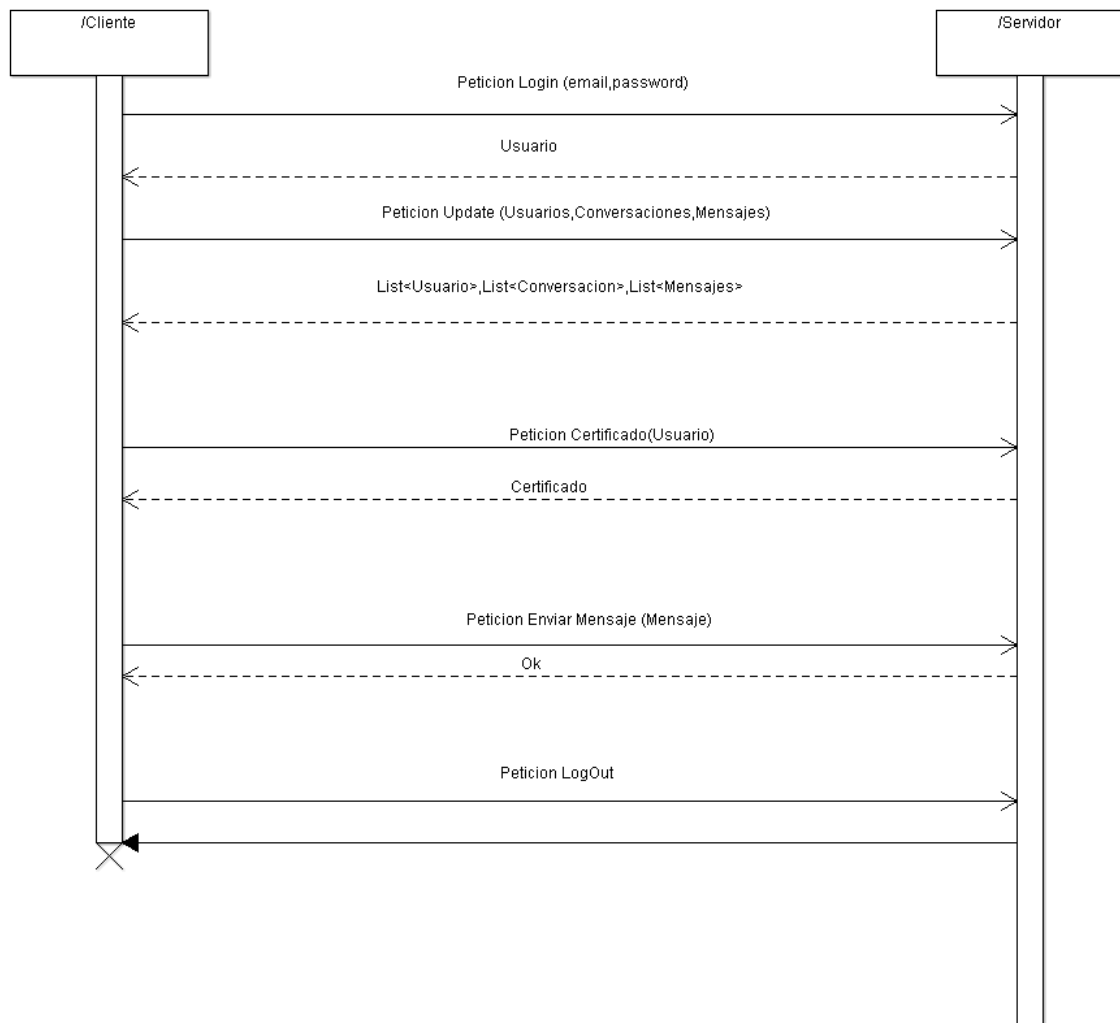
Una vez obtenida la petición genérica, se recupera el valor que indica el tipo concreto que es y se deserializa con la información más profunda:

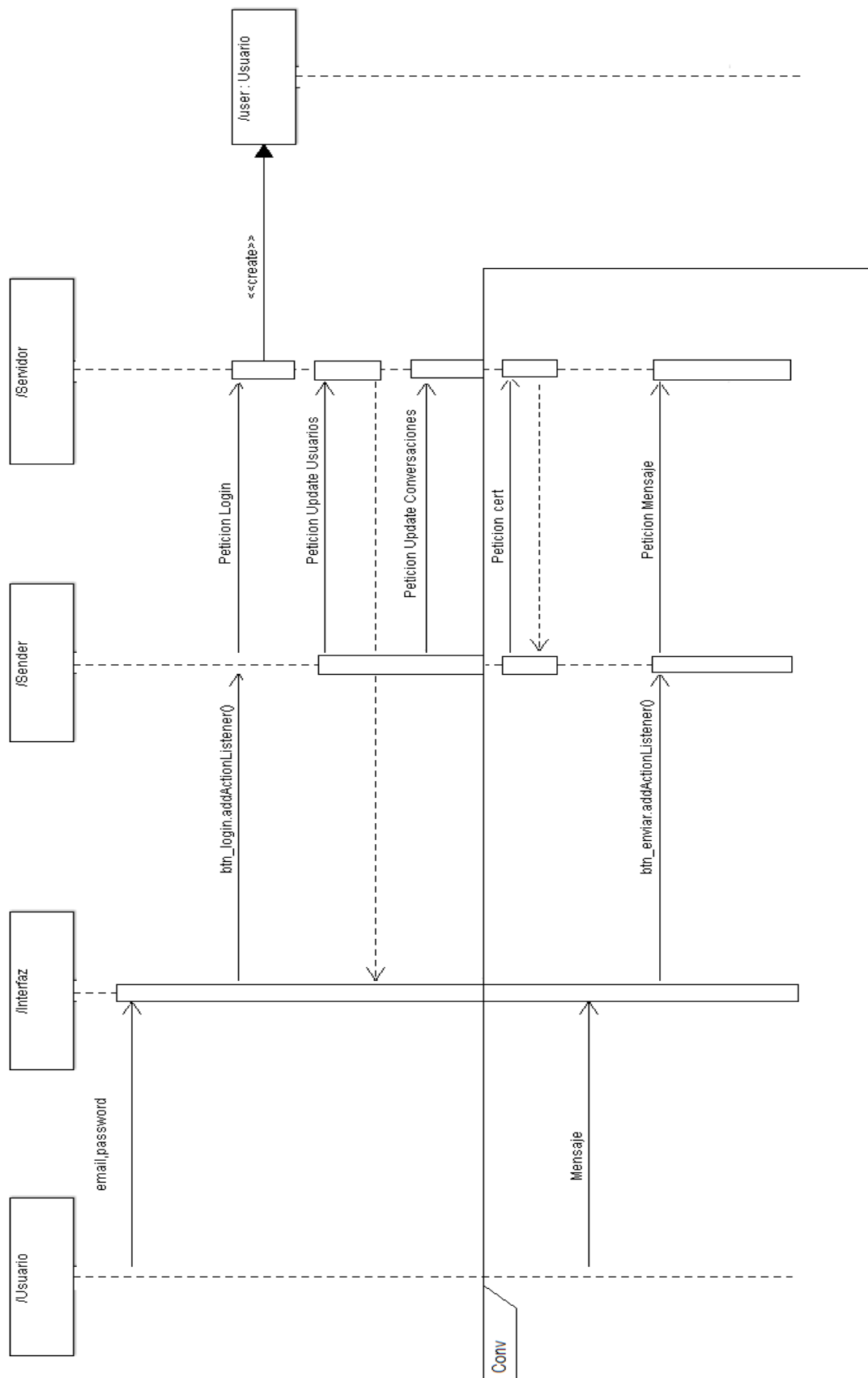
```
PeticionLogin petitiononLOGIN = (PeticionLogin) new
JSONDeserializer<PeticionLogin>().deserialize(obj.toString(),PeticionL
ogin.class);
```

Para esta tarea es necesario tener clases gemelas en ambos lados del entorno, cliente-servidor, además de la exigencia por parte de la librería flexjson de un constructor sin parámetros para su correcto funcionamiento.

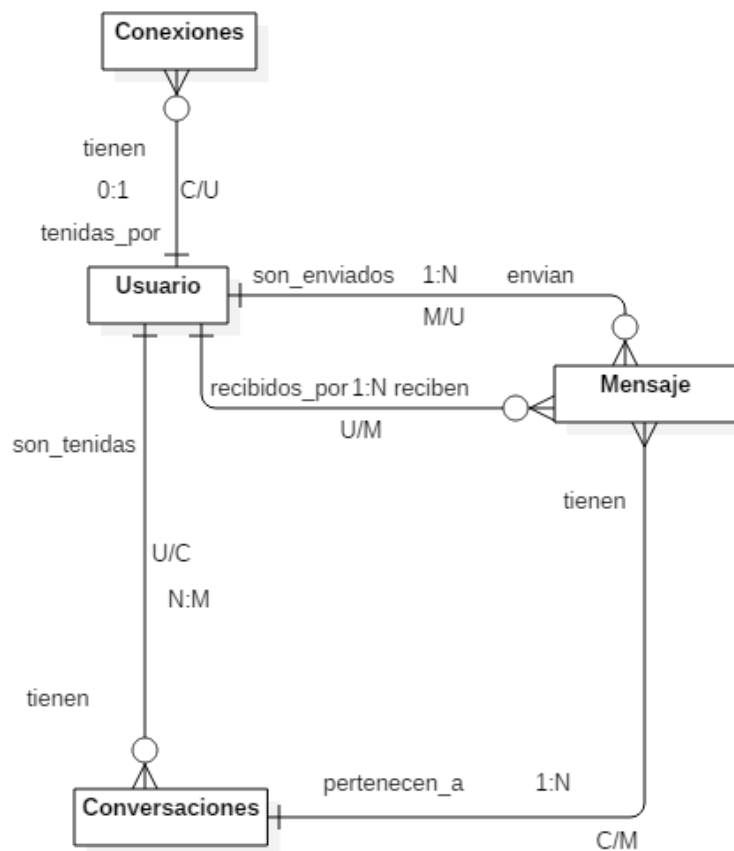
Apuntar también que antes de serializar desde cualquiera de los lados del flujo de información es necesario formatear el json generado quitando cualquier relación que vincule esta información con la clase original. Para esto se utilizan los comodines (*) pasándoselos a un método concreto de la librería (exclude).

```
JSONSerializer serializer = new JSONSerializer();
String ms=serializer.exclude("*.class").serialize(user);
```



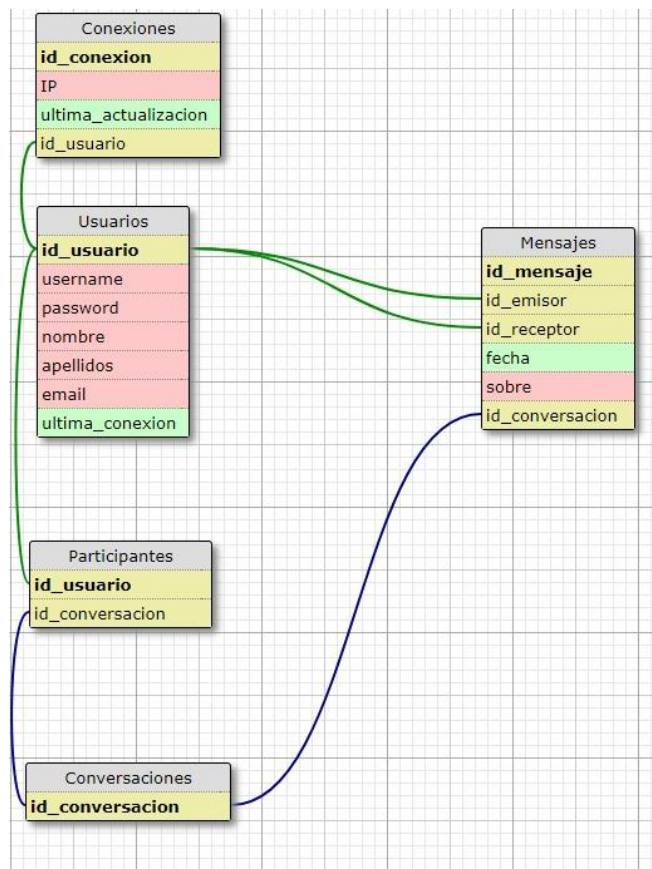


- Modelo de E/R de la base de datos



- La entidad **Usuarios** tiene como atributos el id_usuario, el username, la password, la ultima_actualizacion y el email.
- La entidad **Conexiones** posee como atributos el id_conexion, la ultima_actualizacion, la IP y el id_usuario, que esta referenciado al id_usuario de la entidad Usuarios.
- La entidad **Conversaciones** tiene como atributo la id_conversacion.
- La entidad **Mensaje** tiene como atributos el id_mensaje, el id_emisor y el id_receptor, que están referenciados al id_usuario de la entidad Usuarios, la fecha, el sobre y la id_conversacion, que esta referenciada a la id_conversacion de la entidad Conversaciones.
- La entidad **Participantes** tiene como atributos el id_usuario, que esta referenciado al id_usuario de la entidad Usuario, y la id_conversacion, que esta referenciada a la id_conversacion de la entidad Conversaciones.

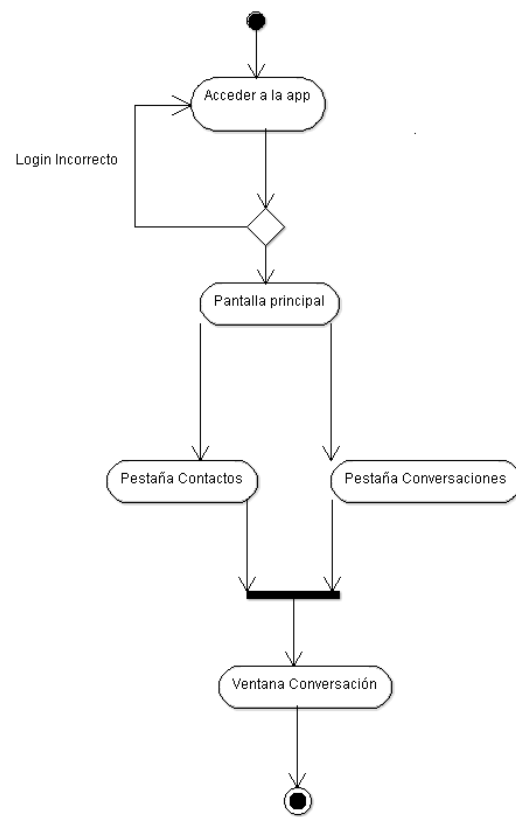
- Diagrama relacional de la base de datos



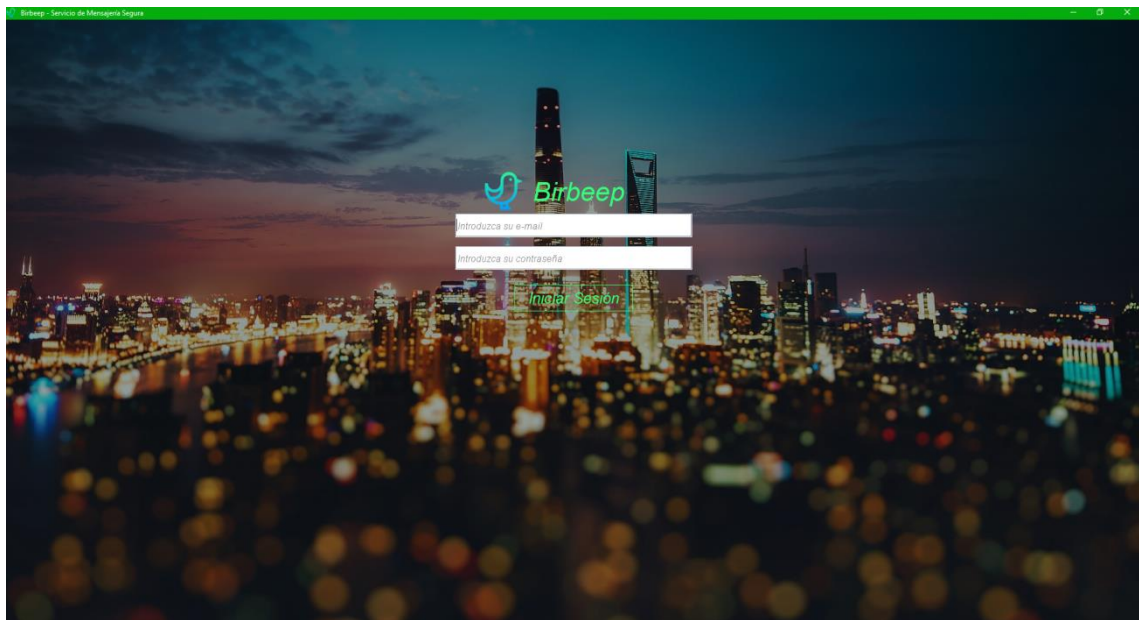
Para usar las funcionalidades de la aplicación, se tendrá que estar registrado en el servidor con un correo y una contraseña.

El servidor almacena en la base de datos los siguientes datos, que son el nombre, los apellidos, la ip desde donde se conecta, una id para cada usuario y el email que le sirve para entrar a la aplicación junto con una contraseña, que también es guardada en la base de datos.

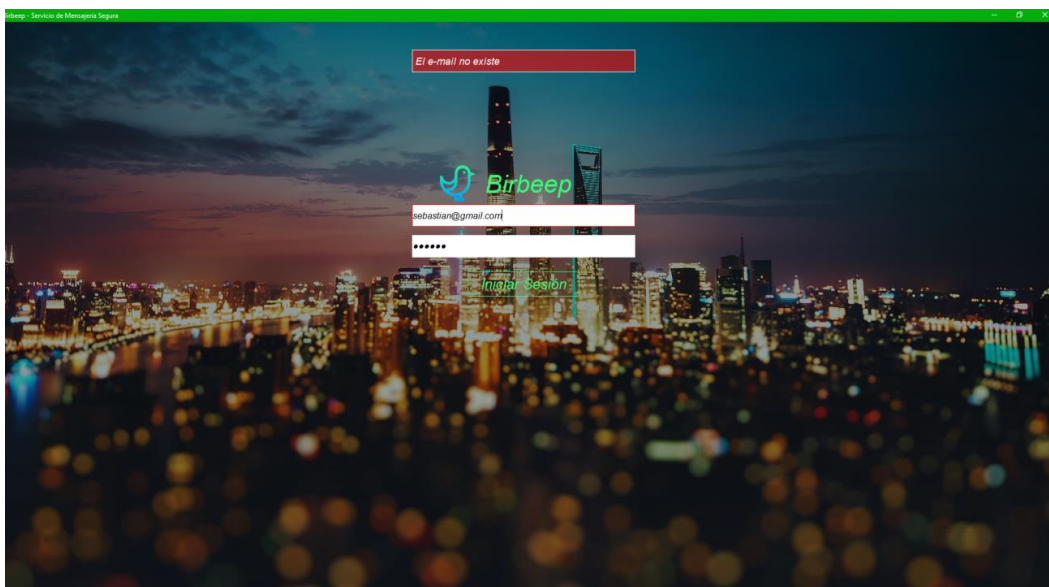
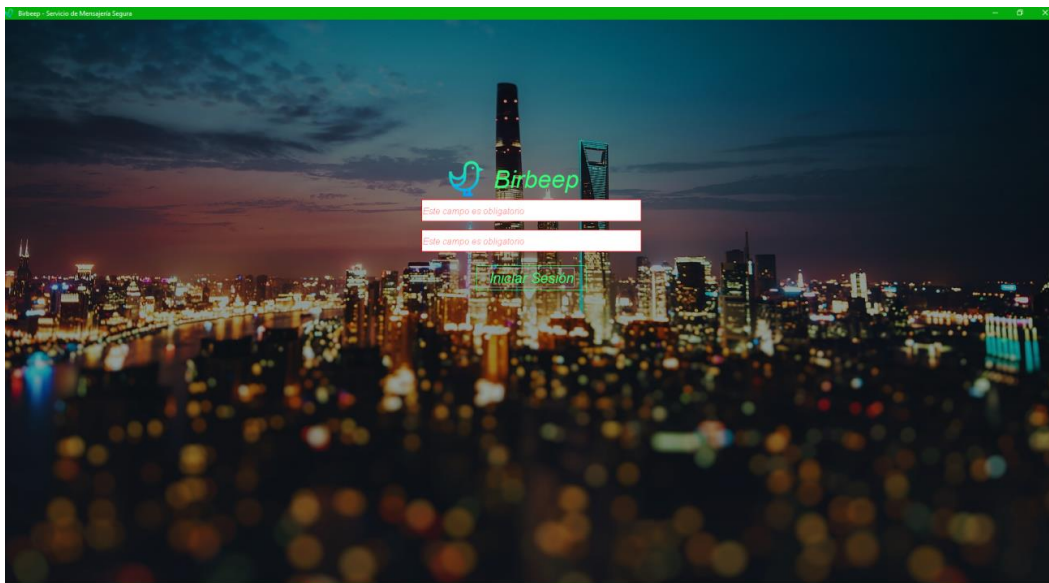
c) Usabilidad



Esta sería la pantalla del inicio de sesión.



En el inicio de sesión se realizan diferentes validaciones.



Esta sería la pantalla principal de la aplicación.



La pestaña **Conversaciones** son las conversaciones que el usuario ya ha tenido con otros usuarios.

La pestaña **Nueva Conversación** son los todos los usuarios dados de alta en la aplicación con los que puedes comenzar una conversación.

En el sector de la derecha de la pantalla cuando se seleccione una conversación en la pestaña **Conversaciones**, o se seleccione un usuario para comenzar una conversación en la pestaña **Nueva Conversación**, aparecerá el chat con el usuario seleccionado. En este chat, aparecen los mensajes enviados y recibidos, si existen en esa conversación, una área de texto junto con un botón para enviar el mensaje.

3. Conclusiones

En este proyecto hemos puesto en práctica gran cantidad de materia vista durante el curso de Desarrollo de Aplicaciones Multiplataforma, aplicando los conocimientos adquiridos.

Dado el tipo de proyecto también hemos debido emplear una extensa labor de investigación y estudio de las tecnologías a utilizar.

Quizás por estas razones se ha puesto de manifiesto la importancia de una base sólida a la hora de afrontar un proyecto y la necesidad, sobre todo en el campo que nos ocupa, de mantener los conocimientos actualizados y no descuidar la formación. El hecho de tratar con un proyecto desde cero y sacarlo adelante supone definir claramente las fases por las que va a evolucionar el mismo para organizar correctamente las tareas a realizar y tener presente las posibles modificaciones que a lo largo del desarrollo puedan surgir. Este desarrollo nos ha servido para ver en primera persona las dificultades y los diferentes retos a los que uno se enfrenta a la hora de abordar una empresa como ésta, en nuestro caso particular también ha servido para conocer mejor cómo se desarrolla el trabajo en equipo, obligándonos a organizarnos lo mejor posible para sacarlo adelante.

4. Recursos

Se han utilizado los equipos personales de los miembros del equipo. No se ha utilizado ningún servidor en la nube, de hecho, incluso la base de datos se ha instalado en el entorno local mediante un paquete que incluye un servidor web y el Sistema Gestor de Bases de Datos con lo que se ha conseguido concentrar todos los recursos en las máquinas personales.

5. Bibliografía

Para la documentación:

[https://es.wikipedia.org/wiki/Eclipse_\(software\)](https://es.wikipedia.org/wiki/Eclipse_(software))

<https://es.wikipedia.org/wiki/PhpMyAdmin>

https://es.wikipedia.org/wiki/M%C3%A1quina_virtual_Java

<https://www.java.com/es/download/help/sysreq.xml>

Para la conexión SSL:

http://chuwiki.chuidiang.org/index.php?title=Socket_SSL_con_Java

Para crear Certificados:

http://chuwiki.chuidiang.org/index.php?title=Encriptacion_con_Java

<https://www.adictosaltrabajo.com/tutoriales/security-ssl-keytool/>

<https://docs.oracle.com/javase/8/docs/technotes/tools/windows/keytool.html>

Consultas generales:

<https://stackoverflow.com/>

<https://docs.oracle.com/javase/8/docs/api/>

Sobres digitales:

<https://docs.oracle.com/javase/tutorial/security/apisign/enhancements.html>

Serialización:

<http://flexjson.sourceforge.net/>

Recursos extra:

Apuntes de la asignatura *Programación de servicios y procesos*:

Tema 3 Los hilos de ejecución

Tema 5 Programación de comunicaciones en red

Tema 7 Programación segura