Práctica Java RMI Middleware

Víctor Portals Lorenzo - R090218

8 de julio de 2014

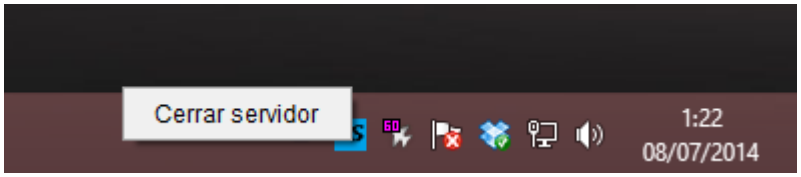
1. Introducción

La aplicación desarrolla implementa algunas de las funcionalidades básicas de una red social, estas son, la publicación de mensajes o estados personales, la inclusión de amigos que puedan ver estos mensajes, y cuyos mensajes se vean reflejados en una sección de “Novedades”. Se trata de una aplicación distribuida desarrollada para un funcionamiento con un servidor y múltiples aplicaciones clientes.

2. Interfaz de usuario

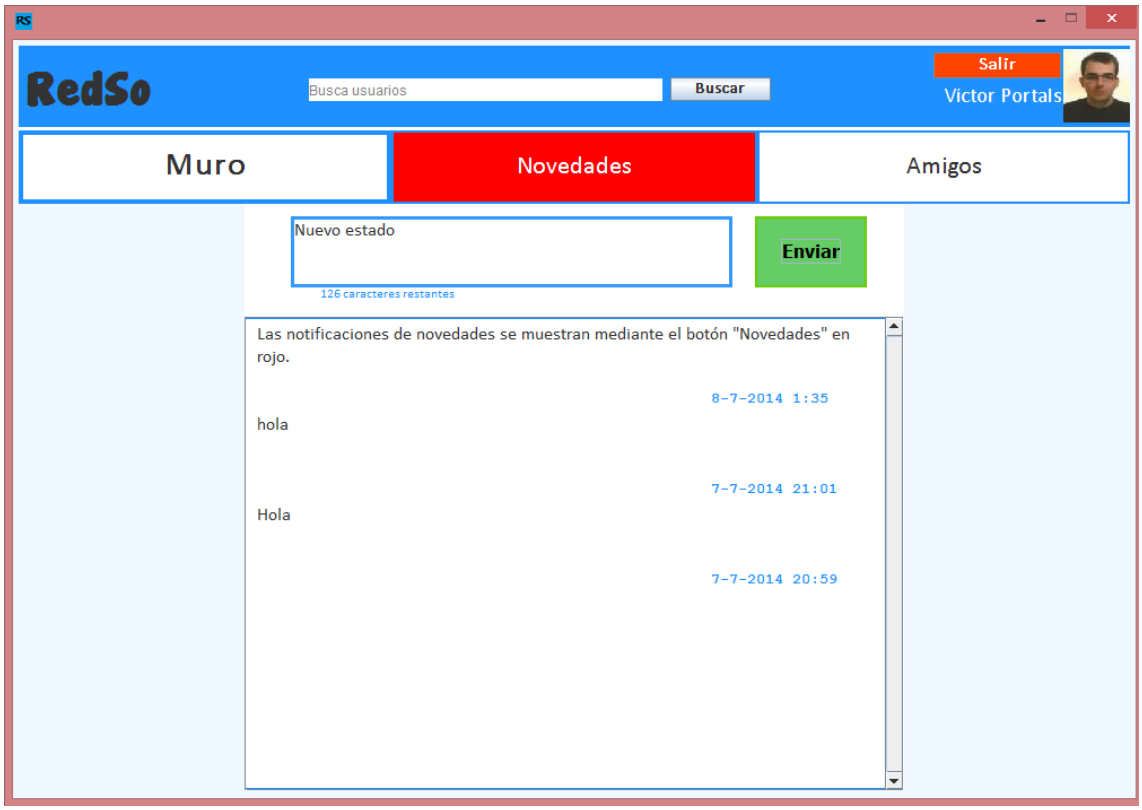
Además de las interfaces remotas de Java RMI descritas en el siguiente apartado la interacción de los usuarios con la aplicación y del administrador con el servidor se realizan mediante interfaces gráficas.

En el caso del servidor se trata únicamente de un icono en el área de notificación que advierte que el programa está en funcionamiento y permite cerrarlo:



Para la aplicación cliente se trata de una GUI de usuario más completa que permite una interacción completa del usuario con el sistema:

La aplicación distribuida tiene notificaciones push mediante callback, estas se ven reflejadas en la interfaz gráfica pintando de rojo la sección afectada:



La interfaz gráfica se ha realizado con las librerías gráficas Java Swing.

Se puede dividir en seis vistas: el muro del usuario, donde se muestran los mensajes de estado enviados por el usuario; el tablón de novedades donde se muestran todos los mensajes, de amigos y propios de más antiguos a más recientes; la sección Amigos, donde se muestran la lista de amigos, incluyendo peticiones pendientes; la vista que muestra los resultados de la búsqueda de usuarios; la zona de perfil, donde se puede editar el mismo y que se accede pulsando sobre la foto de usuario; y por último el muro de un usuario amigo o que tenga la opción de muro público marcada.

3. Interfaces remotas Java RMI.

Servidor:

```
package server;

import java.rmi.Remote;
import java.rmi.RemoteException;

/**
 * Interfaz remota que ofrece una funcionalidad básica previa a la creación de la sesión. Estas
funcionalidades <br>
 * son las de inicio de sesión y registro.
```

```
 * @author Víctor Portals Lorenzo
 *
 */
        public interface ConexionInterfaz extends Remote {
        /**
         * Compueba que el usuario y contraseña sean correctos y crea el objeto remoto Sesion
asociado con el email que recibe como parámetro.
         * Además se encarga de almacenar la dirección IPv4 del usuario para las funciones de
callback.
         * @param email
         * @param pass
         * @return boolean
         * @throws RemoteException
         */
        public boolean iniciarSesion(String email, String pass) throws RemoteException;
        /**
         *  Crea un nuevo objeto User con los parámetros recibidos y llama al método
inciarSesion.
         * @param email
         * @param name
         * @param apellidos
         * @param password
         * @param foto
         * @param web
         * @param publico
         * @return boolean
         * @throws RemoteException
         */
        public boolean crearUsuario(String email, String name, String apellidos, String
password, byte[] foto, String web, boolean publico) throws RemoteException;

}
```

```
package server;
import java.rmi.Remote;
import java.util.List;
import java.rmi.RemoteException;
/**
 * Interfaz remota que abstrae el concepto de sesión, permitiendo al cliente asociado a la
instancia <br>
 * de esta interfaz realizar las funcionalidades básicas de este sistema tras la conexión.
 * @author Víctor Portals Lorenzo
 *
 *
 */
public interface SesionInterfaz extends Remote {

        /**
         * Devuelve una lista iterable de mensaje ordenados por fecha tanto del usuario como de
sus amigos.<br>
         * @param email
         * @return List<Message>
         * @throws RemoteException
         */
        public List<Message> obtenerNovedades(String email) throws RemoteException;
        /**
         * Devuelve un array de Strings que contienen datos del usuario para conformar su muro.
La mayor parte <br>
         * de su funcionalidad a quedado sin uso ya que los mensajes se obtienen por otros
medios.
         * @param email
         * @return String[]
```

```
         * @throws RemoteException
         */
        public String[] getDatos(String email) throws RemoteException;

        /**
         * Devuelve un array de bytes que se corresponde con una foto (de perfil) en formato
jpg.
         * @param email
         * @return byte[]
         * @throws RemoteException
         */
        public byte[] getFoto(String email) throws RemoteException;
        /**
         * En caso que no existir crea un registro de amistad en tabla Friend entre el usuario
que lo invoca y el usuario asociado<br>
         * al email pasado como parámetro. Para no ralentizar el sistema crea un thread paralelo
que se encarga de notificar <br>
         * mediante callback al usuario añadido como amigo sin ralentizar el sistema.
         * @param email_2
         * @throws RemoteException
         */
        public void addFriend(String email_2) throws RemoteException;
        /**
         * El usuario del programa cliente invocante confirma su conformidad con la petición de
amistad del usuario<br>
         * asociado al email pasado como parámetro. Se modifica la tabla Friend.
         * @param email_1
         * @throws RemoteException
         */
        public void confirmFriend(String email_1) throws RemoteException;
        /**
         * Se borra el registro de la tabla Friend que relaciona al usuario invocante y el
usuario asociado al email pasado<br>
         * como parámetro
         * @param email_1
         * @throws RemoteException
         */
        public void refuseFriend(String email_1) throws RemoteException;
        /**
         * Devuelve un array de booleanos que contiene información sobre el registro de dos
usuarios en la tabla Friend. El array <br>
         * devuelto tiene la siguiente estructura [Existe registro , amistad_confirmada ,
invocante_acepta , usuario_parámetro_acepta].
         * @param email_objetivo
         * @return boolean[]
         * @throws RemoteException
         */
        public boolean[] getFriendship(String email_objetivo) throws RemoteException;
        /**
         * Se crea un registro en la tabla Message con los datos del usuario, el mensaje y la
hora al momento de la invocación.<br>
         * Además crea un thread paralelo encargado de notificar a las aplicaciones cliente
mediante callback sin ralentizar el <br>
         * sistema.
         * @param mensaje
         * @throws RemoteException
         */
        public void publishMessage(String mensaje) throws RemoteException;
        /**
         * Devuelve una lista iterable de objetos Message cuyo email_ID se corresponda con el
pasado como parámetro.
         * @param email
         * @return List\<Message\>
         * @throws RemoteException
```

```
         */
        public List<Message> getMessages(String email) throws RemoteException;
        /**
         * Devueve una lista iterable de objetos User que cumplan con el siguiente criterio de
busqueda: <br>
         * Se separa el String recibido como parámetro en palabras, y se compara cada una con el
campo email y con el campo apellidos <br>
         * en busca de coincidencias.
         * @param busqueda
         * @return List\<User\>
         * @throws RemoteException
         */
        public List<User> getResults(String busqueda) throws RemoteException;
        /**
         * Devueve una lista iterable de objetos User cuyo atributo email sea el campo email_2
donde el atributo email del <br>
         * usuario invocante sea el campo email_1 en la tabla Friend o viceversa.
         * en busca de coincidencias.
         * @param email
         * @return List\<User\>
         * @throws RemoteException
         */
        public List<User> getFriends(String email) throws RemoteException;
        /**
         * Elimina el objeto remoto asociado con un usuario que ha inicado sesión.
         * @param email
         * @throws RemoteException
         */
        public void logOut(String email) throws RemoteException;

        public boolean getPublic(String email) throws RemoteException;

        public boolean modifyProfile(String nombre, String apellidos, String web, Boolean
isPublic, byte[] foto) throws RemoteException;
}
```

Cliente:

```
package newclient;

import java.rmi.Remote;
import java.rmi.RemoteException;
/**
 * Interfaz remota que ofrece la funcionalidad callback para el servidor de la aplicación,
permitiendo que esté <br>
 * notificar las novedades y las nuevas solicitudes de amistad.
 * @author Víctor Portals Lorenzo
 *
 */
public interface PushNewsInterface extends Remote {

        /**
         * Cambia el color del botón de novedades (btnNovedades) declarado como un atributo de
clase en la clase <br>
         * MainClient.java. Además da el valor true al flag hayNovedad (también atributo de
clase en MainClient.java)<br>
         * que evitará que la notificación desaparezca hasta que no se acceda a la sección de
Novedades.
         * @param email
         * @return boolean
         * @throws RemoteException
         */
```

```
        public boolean recibirNotificacion(String email) throws RemoteException;
        /**
          * Cambia el color del botón de amigos (btnAmigos) declarado como un atributo de clase
en la clase <br>
         * MainClient.java. Además da el valor true al flag haySolicitud (también atributo de
clase en MainClient.java)<br>
         * que evitará que la notificación desaparezca hasta que no se acceda a la sección de
Amigos.
         * @param email
         * @return boolean
         * @throws RemoteException
         */
        public boolean nuevaSolicitud(String email) throws RemoteException;

        public String consultaIdentidad() throws RemoteException;

}
```

4. Diagrama de clases del sistema distribuido

Diagrama del cliente:

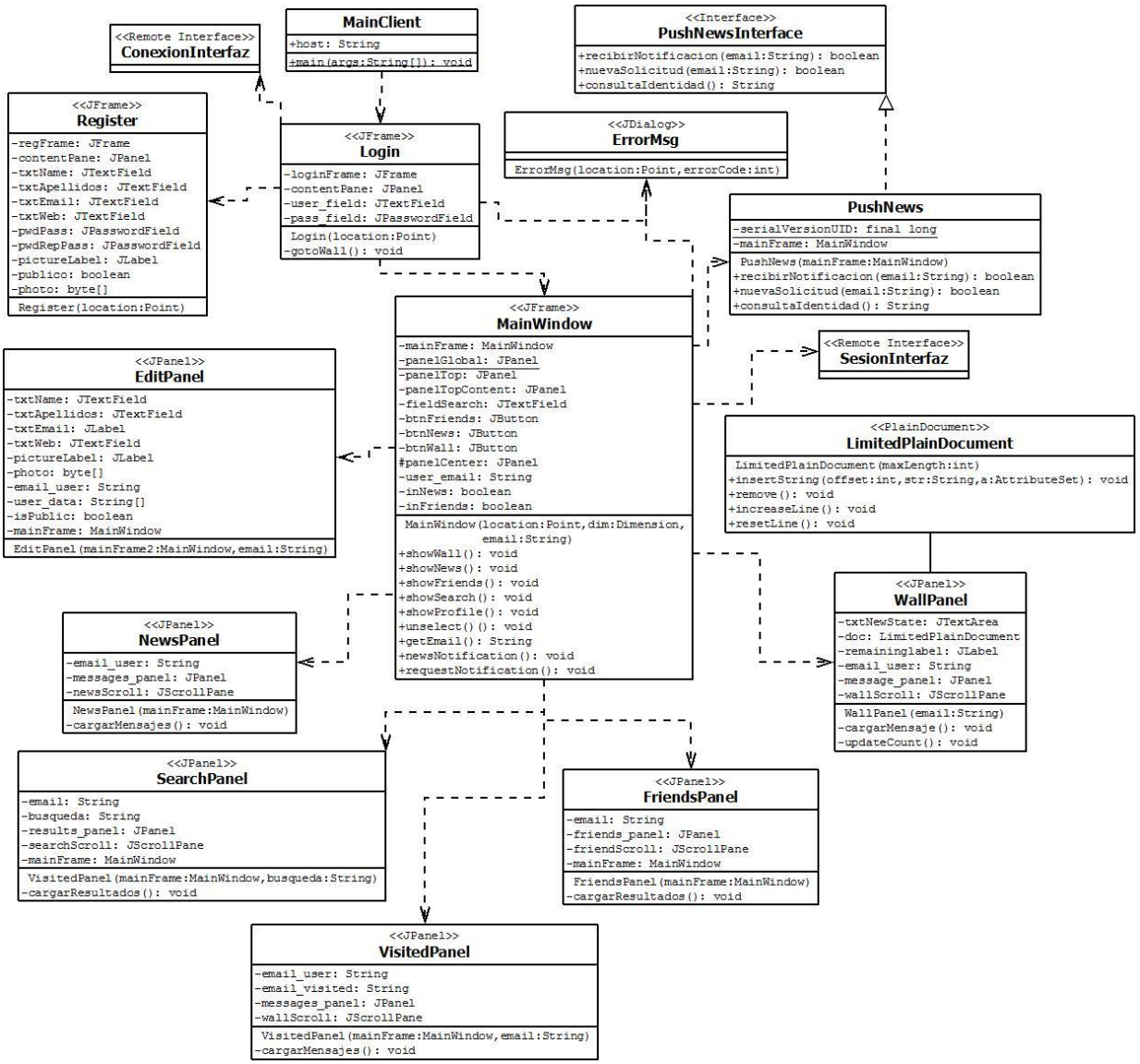
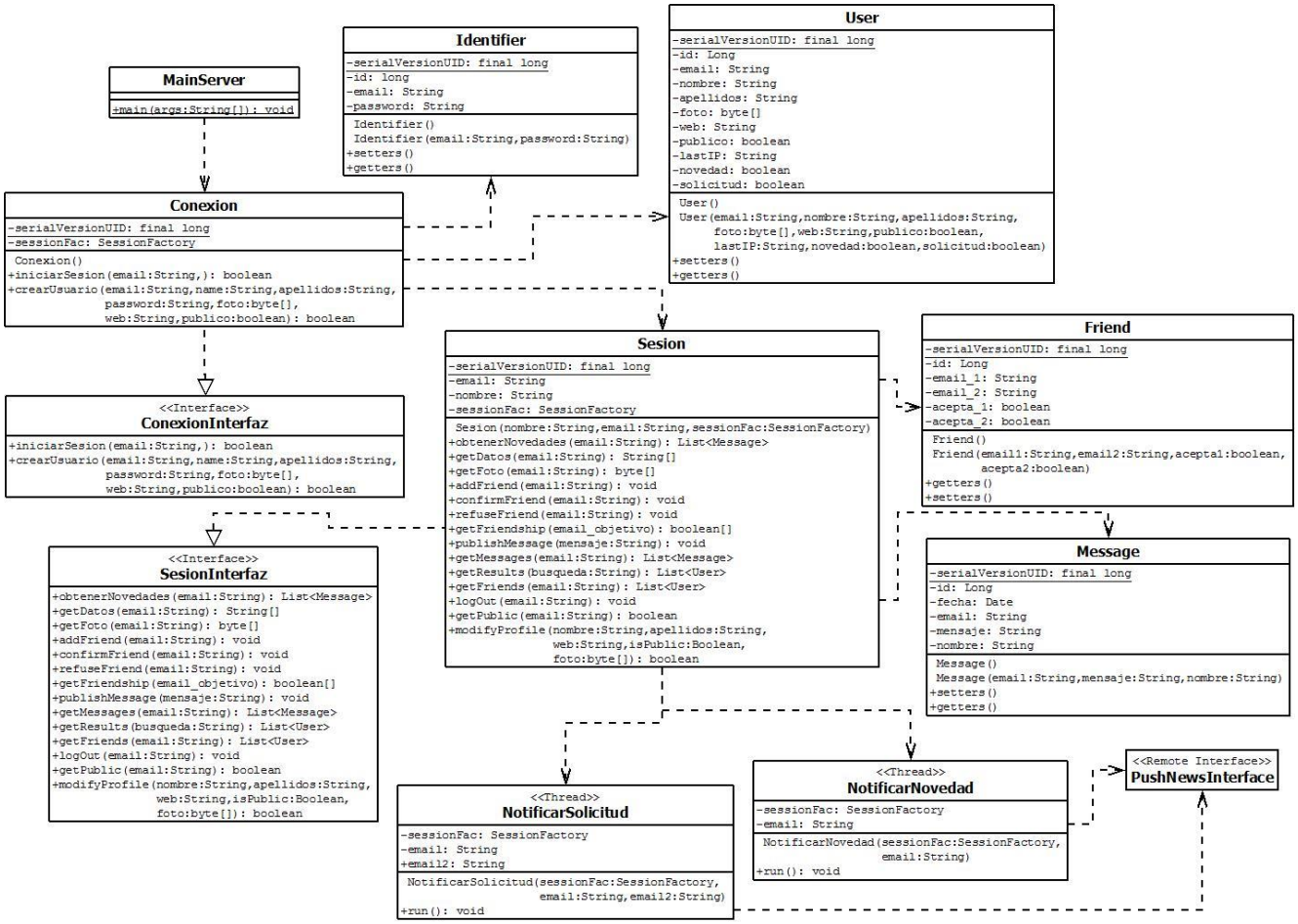


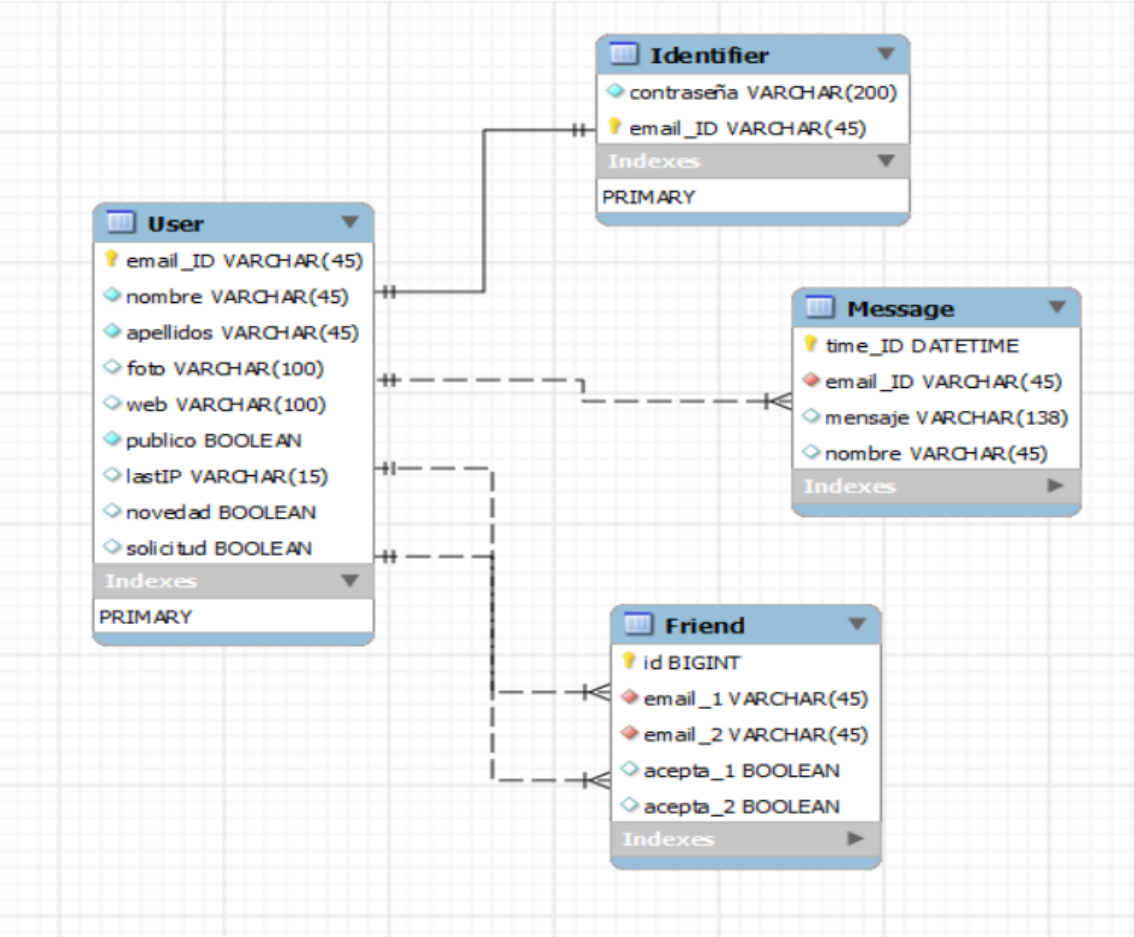
Diagrama del servidor:



5. Persistencia en el servidor

La persistencia de los datos en el servidor se ha implementado mediante la herramienta de mapeo de objetos Hibernate y un servidor MySQL 5.6. Se han creado cuatro tablas, por el funcionamiento de Hibernate las tablas reciben el mismo nombre que los objetos que mapean:

- User: guarda los datos de personales de los usuarios (nombre, apellidos, email, privacidad…).
- Identifier: guarda las contraseñas de los usuarios de forma independiente para evitar la manipulación innecesaria de esta información delicada. Las contraseñas se cifran en la aplicación cliente mediante el algoritmo irreversible hash SHA-256.
- Friend: guarda la relación entre usuarios y el grado de aceptación de los mismos.
- Message: guarda los mensajes de todos los usuarios asociados al email de los mismos.



6. Conclusiones y opinión personal

Personalmente me ha parecido un trabajo muy interesante. Me ha gustado trabajar desarrollando una aplicación con interfaz gráfica, he conocido herramientas muy interesantes como Hibernate y también me ha ayudado a comprender el funcionamiento de aplicaciones distribuidas.

Además de aprender sobre el propio campo del middleware también me ha ayudado a repasar conceptos de otras áreas de la programación y otras materias.