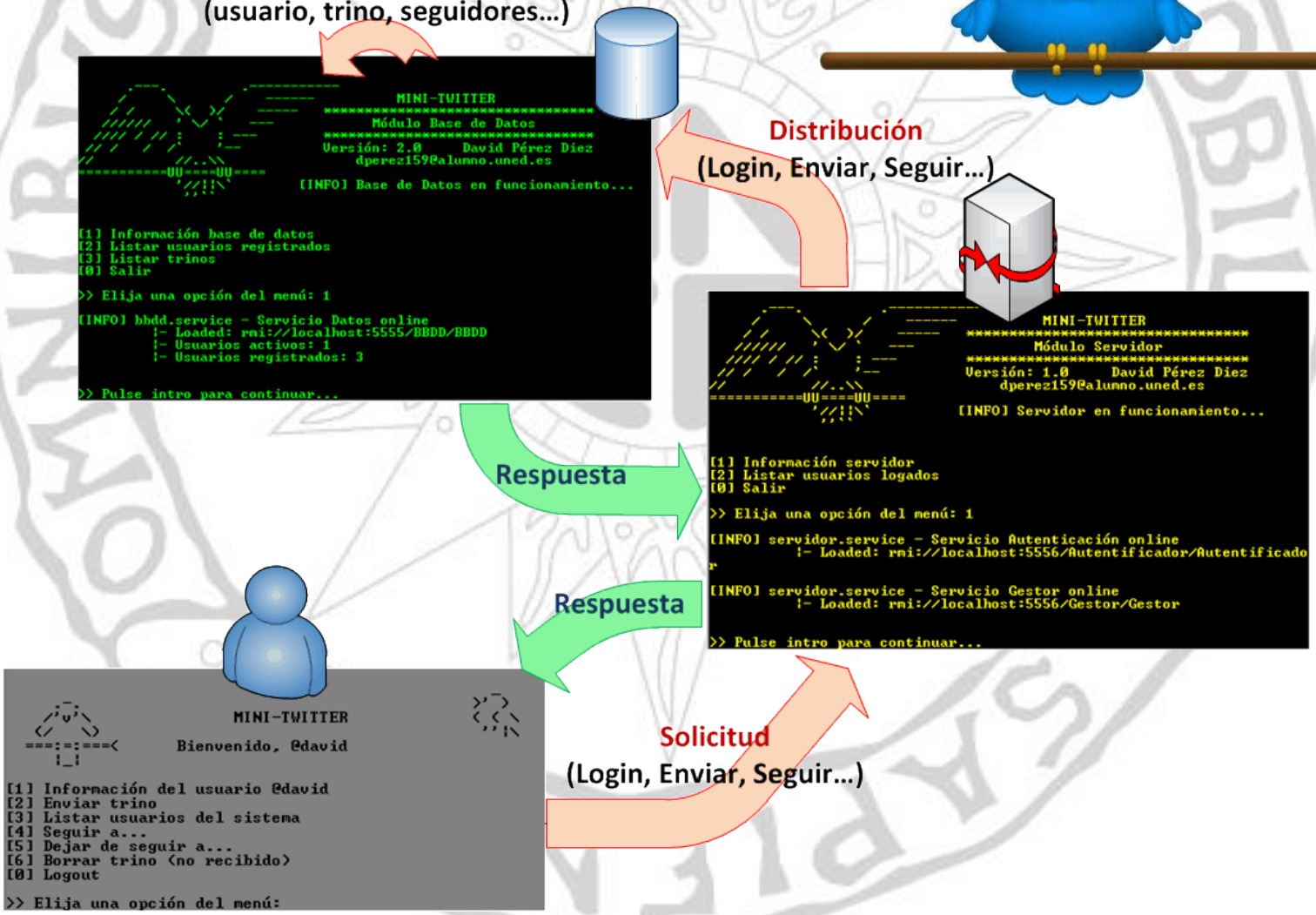
Práctica de Sistemas Distribuidos



Alumno: David Pérez Diez
Teléfono:
DNI:
CA: Estudios Fiscales, Madrid
E-mail: dperez159@alumno.uned.es

Curso: 2018-2019

Contenido

➢ CONSIDERACIONES GENERALES

Entorno de la práctica

La práctica emula a un sistema básico de microblogging al estilo del famoso Twitter® usando Java RMI y está desarrollada en su totalidad sobre **plataforma Microsoft**.

El IDE utilizado es **NetBeans IDE 8.1** bajo *Windows 10 x64* con las siguientes especificaciones:

```
Product Version: NetBeans IDE 8.1 (Build 201510222201)
Updates: Updates available to version NetBeans 8.1 Patch 1
Java: 1.8.0_144; Java HotSpot(TM) 64-Bit Server VM 25.144-b01
Runtime: Java(TM) SE Runtime Environment 1.8.0_144-b01
```

Ilustración 1 - Información del entorno de codificación

Mientras que el escenario de pruebas se ha realizado en un *Windows 7 Enterprise x64* y el software **JDK v11** de Oracle:

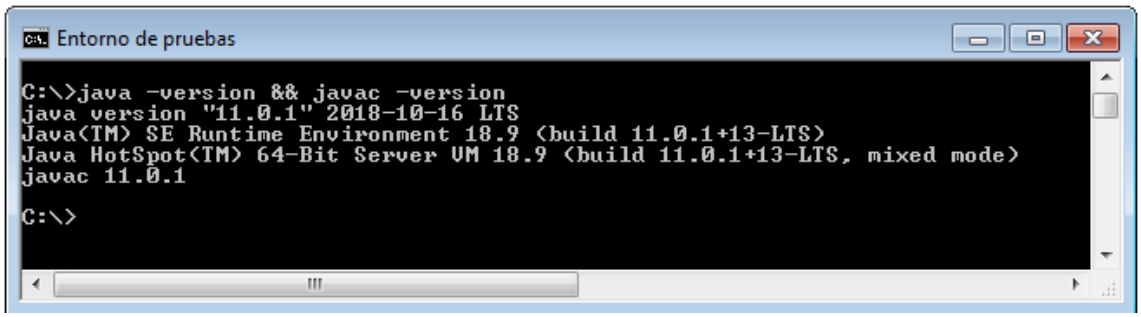


Ilustración 2 – Información del entorno de pruebas

El motivo de tener **dos entornos** distintos (codificación y pruebas) es debido a la incompatibilidad entre versiones de IDE y JDK utilizados.

Se han creado **tres proyectos** por cada módulo especificado en el enunciado de la práctica, más un **proyecto común** para los tres.

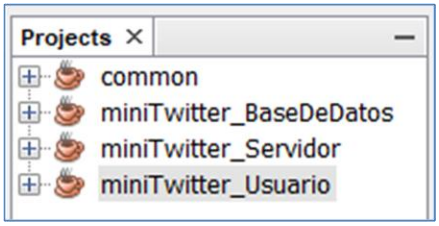


Ilustración 3 - Proyectos

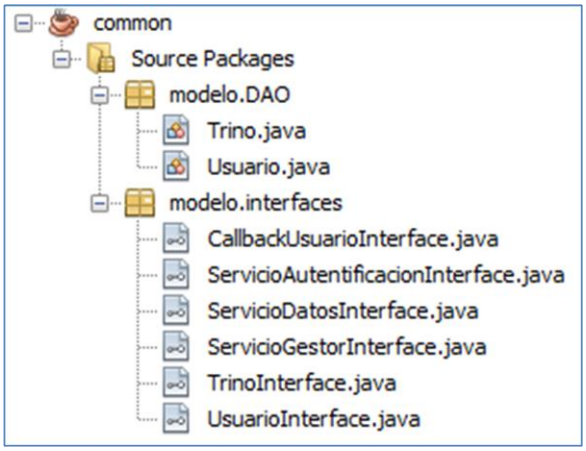
Todas las clases e interfaces de cada proyecto están comentadas mediante **JavaDoc**.

Estructura de la práctica

Se ha intentado seguir en la medida de lo posible el patrón de arquitectura de software de **tres componentes**, *Modelo-Vista-Controlador (***MVC***)*, estableciendo divisiones y responsabilidades concretas en cada una de ellas.

La funcionalidad es que el usuario solicita peticiones al servidor y éste se las solicita a la base de datos, siendo el **servidor la parte intermedia** entre usuario y base de datos, no existiendo comunicación directa entre usuario y base de datos.

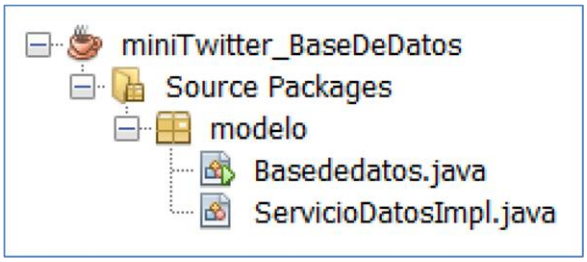
La estructura junto con sus acciones es la siguiente:



Proyecto `common`, contiene el código común y necesario para la intercomunicación de los otros tres proyectos, entre ellos todas las interfaces del resto de clases.

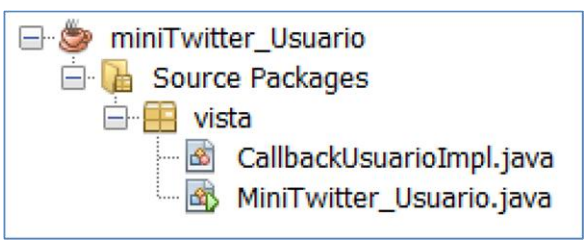
En `modelo.DAO` se encuentra los objetos que contienen los datos, como la clase `Trino.java` proporcionada por el Equipo Docente y `Usuario.java`.

Mientras que `modelo.interfaces` contiene las interfaces que tienen que conocer y usar las clases de los distintos proyectos.



Proyecto `Base de datos` es el **modelo**, es decir, donde se engloba la base de datos. Define cómo se comporta la aplicación y responde a las peticiones del **controlador**, guardando los datos en sus estructuras de datos.

Proyecto `Servidor` es el **controlador**, es decir, el intermediario. Es quien lleva el flujo del programa sabiendo quién y cómo se deben de realizar las respectivas acciones que le solicita la **vista** (usuario).



Proyecto `Usuario` es la **vista**, es quien interactúa con el usuario final, es decir, recoge datos del usuario y los muestra formateados.

Las tres interfaces de usuario (base de datos, servidor y usuario) estás diseñadas para uso con **terminal** o símbolo de sistema.

Se ha intentado aplicar una **programación defensiva** para garantizar el comportamiento de las interfaces ante el uso incorrecto o imprevisible por parte del usuario donde hay cuatro tipos de mensajes:

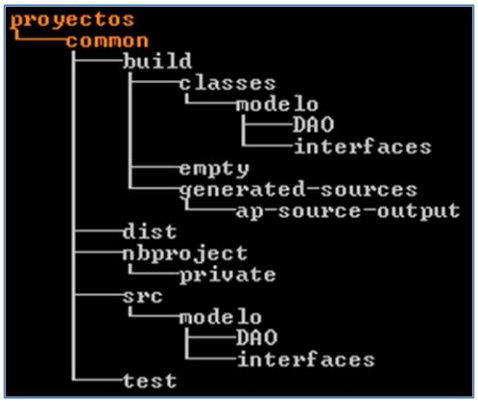
- **[INFO]:** información referente al sistema, como un listado de resultados, sea usuarios logados, usuarios a los que se sigue o información de un módulo en concreto.
- **[OK]:** cuando la operación se ha realizado correctamente, sea dar de alta un usuario, seguir a un usuario o dejar de seguir.
- **[KO]:** cuando la operación no se ha completado satisfactoriamente, como seguir a un usuario que no existe o dar de alta a un usuario con un alias/Nick ya existente.
- **[FAILED]:** excepciones del aplicativo como por ejemplo error al acceder al registro

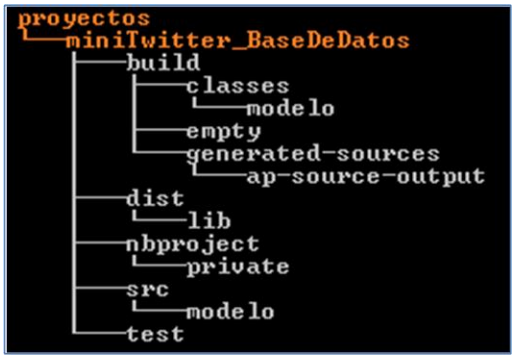
Al no ser un requisito **no existe persistencia**, es decir, cada vez que se finalice la ejecución de los respectivos módulos lo datos almacenados en el programa se perderán.

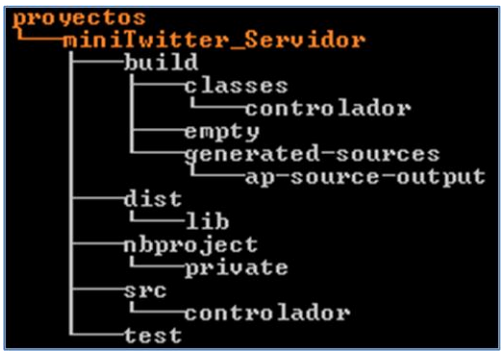
Fuentes y ejecutables

Los fuentes y ejecutables de la práctica están organizados de la siguiente manera:

- **Proyectos:** contiene los cuatro proyectos (`common`, `miniTwitter_BaseDeDatos`, `miniTwitter_Servidor` y `miniTwitter_Usuario`) junto con sus ficheros fuente (.java) y ejecución (.class).







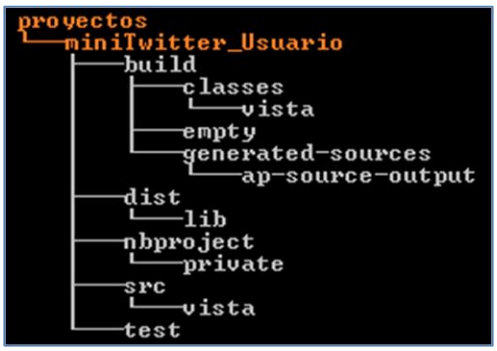


Tabla 1 - Jerarquía del proyecto

- **Ejecutables**: contiene los ficheros .jar y los .cmd para poder realizar la ejecución de forma ordenada y correcta, explicado en el punto **Uso de la práctica**.

Tabla 2 - Jerarquía de ejecutables

He considerado esta estructura, porque creo que es más cómoda de navegar y por lo tanto de evaluar.

Uso de la práctica

Las comunicaciones entre los respectivos módulos (base de datos, servidor y usuario) se realizan por el puerto **5555** y **5556**, por lo que no debiera de haber ninguna otra aplicación haciendo uso de ellos.

Las direcciones o los objetos remotos informados mediante una URL hacen uso de la dirección de **localhost**.

El **orden de ejecución** correcto es primero el módulo `miniTwitter_BaseDeDatos`, seguido de `miniTwitter_Servidor` y por último `miniTwitter_Usuario`. Cualquier otra opción devolverá un error controlado informando de la descripción del problema.
Para facilitar la tarea se proporciona un pequeño script encargado de dichas acciones, `arranque_controlado.cmd`.

Referente al punto anterior, ante errores o llamadas a la opción **Salir/Logout** se finaliza con un `System.exit(0);`

Al arrancar los diferentes módulos existe la posibilidad de que se muestre una **alerta del firewall** de Windows, el cual debemos permitir el acceso para el correcto funcionamiento de la aplicación.

Ilustración 4 - Excepción regla FW

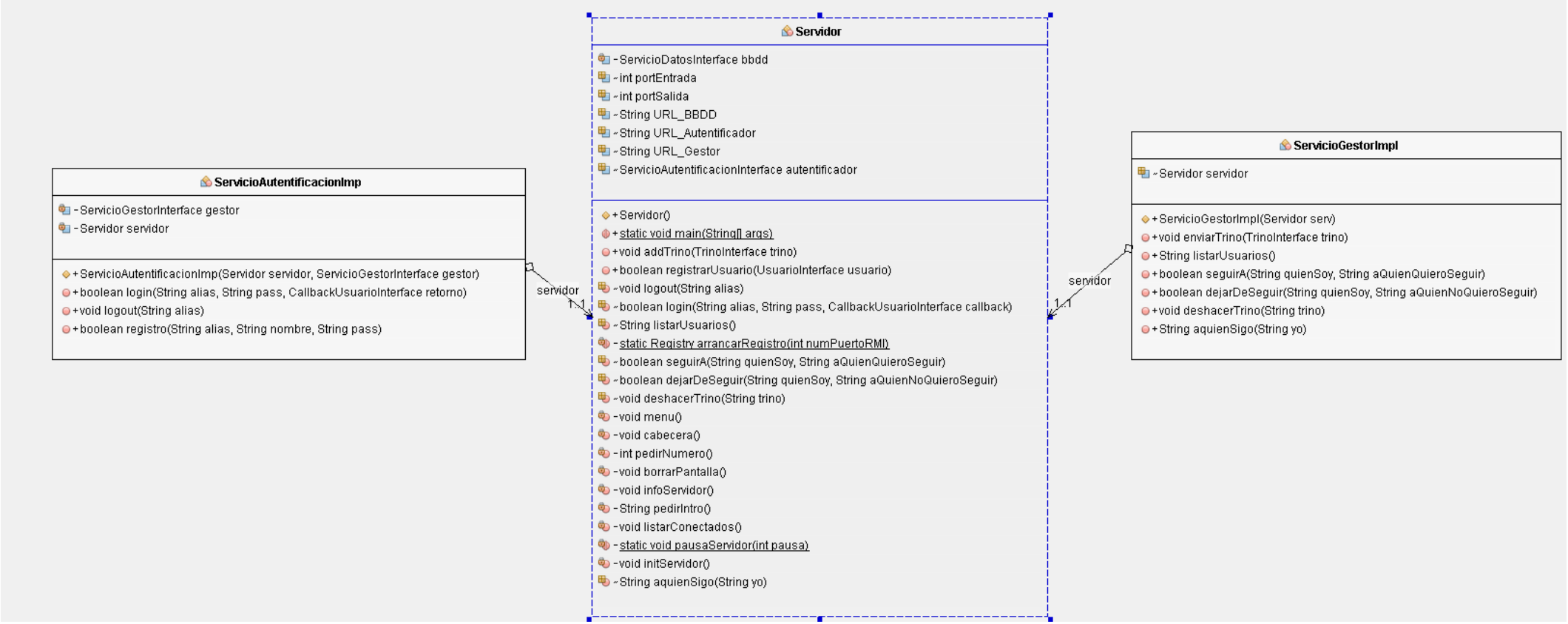
Se pueden ejecutar tantos módulos de usuario como se quiera, pero sólo un módulo por servidor y por base de datos, ya que estos últimos son los que registran los servicios necesarios en el sistema.

Se facilita en este mismo documento un apartado donde se detalla un **FUNCIONAMIENTO DEL SISTEMA** con las acciones más notables del aplicativo, más un video publicado en **YOUTUBE** con una batería de pruebas más exhaustiva que la del ejemplo, para facilitar en la medida de lo posible la corrección al evaluador.

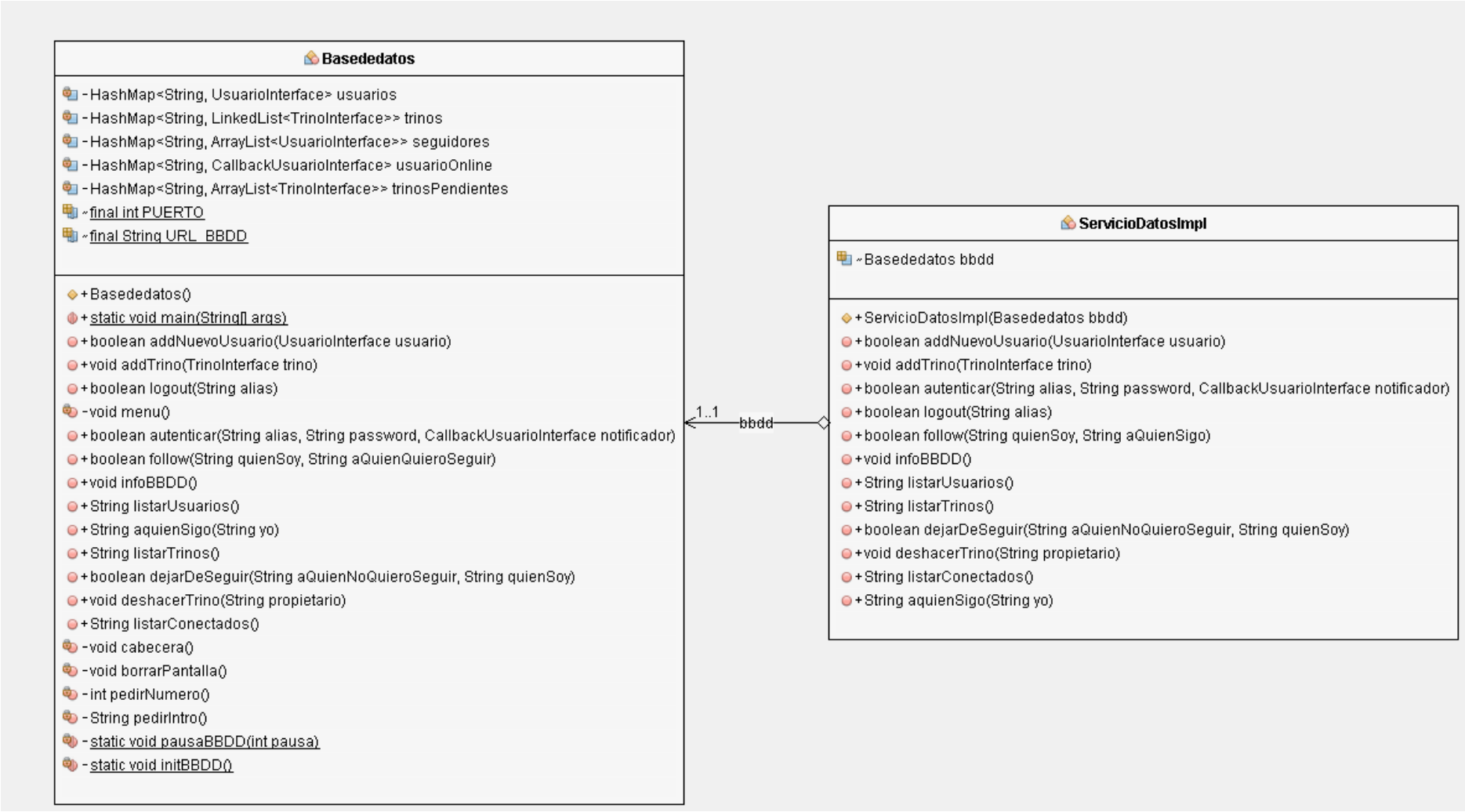
➢ DIAGRAMA DE CLASES

Usuario

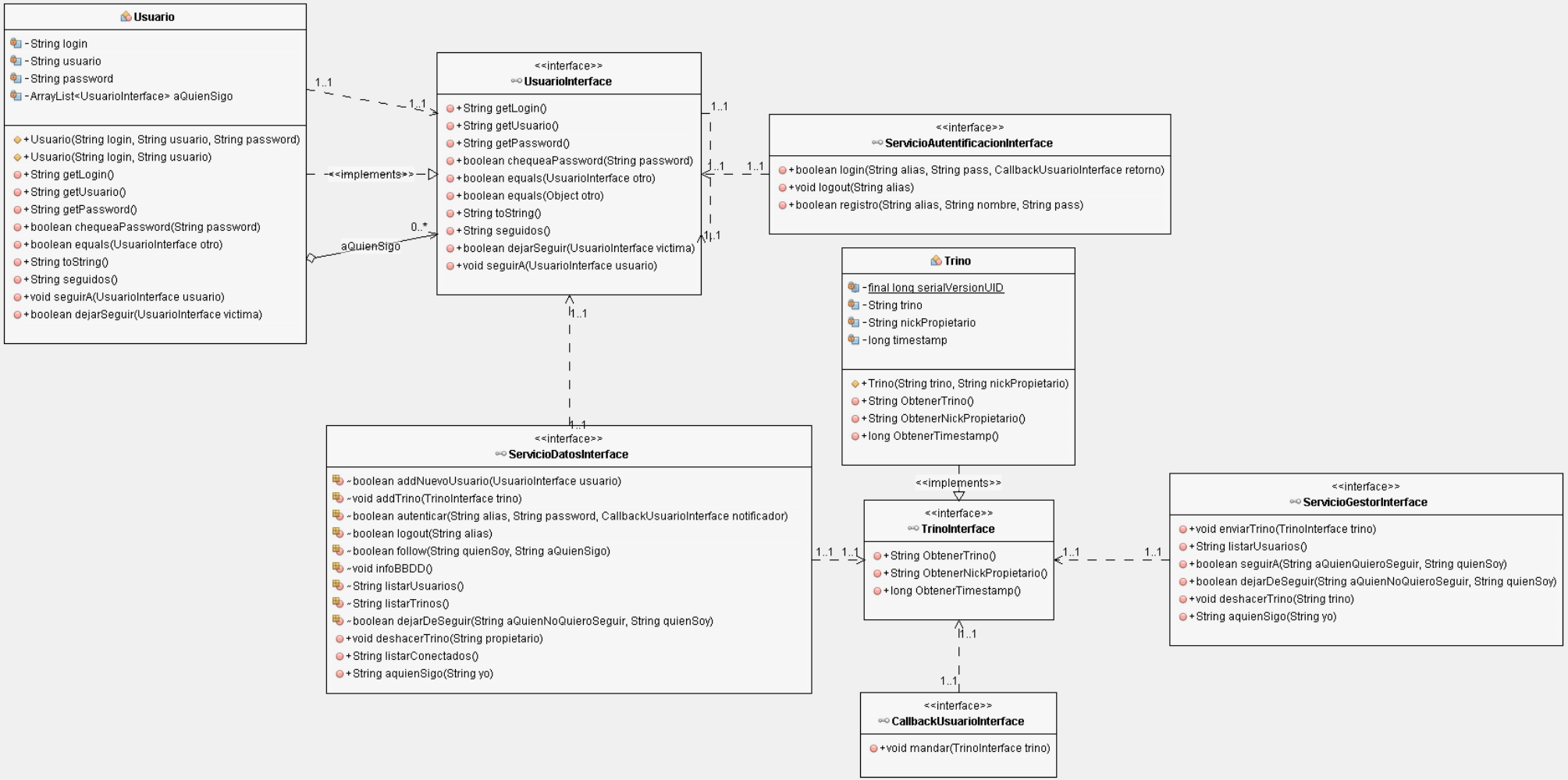
Servidor


ServicioAutentificacionImp
- ServicioGestorInterface gestor
- Servidor servidor
+ ServicioAutentificacionImp(Servidor servidor, ServicioGestorInterface gestor)
+ boolean login(String alias, String pass, CallbackUsuarioInterface retorno)
+ void logout(String alias)
+ boolean registro(String alias, String nombre, String pass)
servidor
1..1
Servidor
- ServicioDatosInterface bbdd
~ int portEntrada
~ int portSalida
~ String URL_BBDD
~ String URL_Autentificador
~ String URL_Gestor
~ ServicioAutentificacionInterface autentificador
+ Servidor()
+ static void main(String[] args)
+ void addTrino(TrinoInterface trino)
+ boolean registrarUsuario(UsuarioInterface usuario)
~ void logout(String alias)
~ boolean login(String alias, String pass, CallbackUsuarioInterface callback)
~ String listarUsuarios()
- static Registry arrancarRegistro(int numPuertoRMI)
~ boolean seguirA(String quienSoy, String aQuienQuieroSeguir)
~ boolean dejarDeSeguir(String quienSoy, String aQuienNoQuieroSeguir)
~ void deshacerTrino(String trino)
- void menu()
- void cabecera()
- int pedirNumero()
- void borrarPantalla()
- void infoServidor()
- String pedirIntro()
- void listarConectados()
- static void pausaServidor(int pausa)
- void initServidor()
~ String aquienSigo(String yo)
servidor
1..1
ServicioGestorImpl
~ Servidor servidor
+ ServicioGestorImpl(Servidor serv)
+ void enviarTrino(TrinoInterface trino)
+ String listarUsuarios()
+ boolean seguirA(String quienSoy, String aQuienQuieroSeguir)
+ boolean dejarDeSeguir(String quienSoy, String aQuienNoQuieroSeguir)
+ void deshacerTrino(String trino)
+ String aquienSigo(String yo)

Base de datos



Common


Usuario
- String login
- String usuario
- String password
- ArrayList<UsuarioInterface> aQuienSigo
+ Usuario(String login, String usuario, String password)
+ Usuario(String login, String usuario)
+ String getLogin()
+ String getUsuario()
+ String getPassword()
+ boolean chequeaPassword(String password)
+ boolean equals(UsuarioInterface otro)
+ String toString()
+ String seguidos()
+ void seguirA(UsuarioInterface usuario)
+ boolean dejarSeguir(UsuarioInterface victima)
<<implements>>
aQuienSigo
0..*
1..1
<<interface>>
UsuarioInterface
+ String getLogin()
+ String getUsuario()
+ String getPassword()
+ boolean chequeaPassword(String password)
+ boolean equals(UsuarioInterface otro)
+ boolean equals(Object otro)
+ String toString()
+ String seguidos()
+ boolean dejarSeguir(UsuarioInterface victima)
+ void seguirA(UsuarioInterface usuario)
<<interface>>
ServicioAutentificacionInterface
+ boolean login(String alias, String pass, CallbackUsuarioInterface retorno)
+ void logout(String alias)
+ boolean registro(String alias, String nombre, String pass)
Trino
- final long serialVersionUID
- String trino
- String nickPropietario
- long timestamp
+ Trino(String trino, String nickPropietario)
+ String ObtenerTrino()
+ String ObtenerNickPropietario()
+ long ObtenerTimestamp()
<<implements>>
<<interface>>
ServicioDatosInterface
~ boolean addNuevoUsuario(UsuarioInterface usuario)
~ void addTrino(TrinoInterface trino)
~ boolean autenticar(String alias, String password, CallbackUsuarioInterface notificador)
~ boolean logout(String alias)
~ boolean follow(String quienSoy, String aQuienSigo)
~ void infoBBDD()
~ String listarUsuarios()
~ String listarTrinos()
~ boolean dejarDeSeguir(String aQuienNoQuieroSeguir, String quienSoy)
+ void deshacerTrino(String propietario)
+ String listarConectados()
+ String aquienSigo(String yo)
<<interface>>
TrinoInterface
+ String ObtenerTrino()
+ String ObtenerNickPropietario()
+ long ObtenerTimestamp()
<<interface>>
ServicioGestorInterface
+ void enviarTrino(TrinoInterface trino)
+ String listarUsuarios()
+ boolean seguirA(String aQuienQuieroSeguir, String quienSoy)
+ boolean dejarDeSeguir(String aQuienNoQuieroSeguir, String quienSoy)
+ void deshacerTrino(String trino)
+ String aquienSigo(String yo)
<<interface>>
CallbackUsuarioInterface
+ void mandar(TrinoInterface trino)

➢ PLANTEAMIENTO DE LA PRÁCTICA

Siguiendo el enunciado de la práctica se han desarrollado las capacidades de los tres actores del sistema, que son usuario, servidor y base de datos.

El usuario se comunica con el servidor mediante dos servicios: **Autenticación** (para registro y/o login) y **Gestor** (para el resto de las acciones como son enviar trino, seguir a, dejar de seguir a…), mientras que el servidor se comunica con la base de datos vía servicio **Datos**.

Una forma gráfica de representar este último punto es el siguiente:

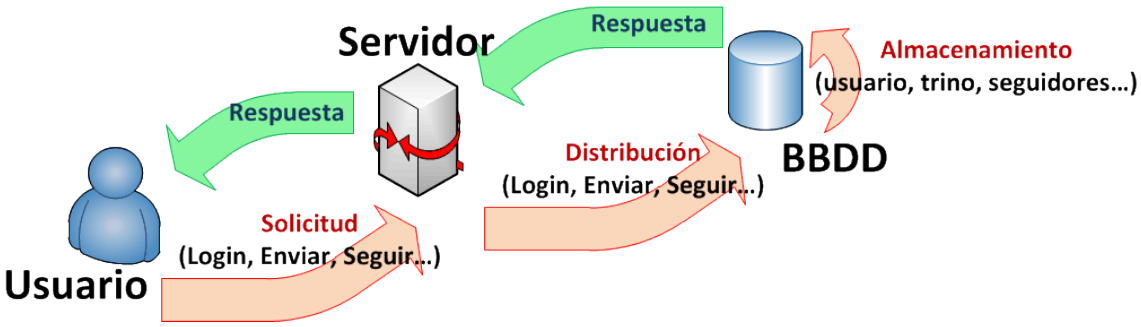


Ilustración 5 - Secuencia de petición-respuesta

En un caso práctico el usuario solicita una tarea al módulo servidor, por ejemplo, **login** (vía Serv. Autenticación) y tras haber sido aceptado el acceso, se le permite el uso del Serv. Gestor, para poder mandar un trino.

El servidor una vez recibe la primera solicitud, **login**, proporciona la petición al módulo de base de datos (vía Serv. Datos), y decide si el usuario puede logar, si es correcto gestionar la siguiente petición, enviar trino.

Dicho de otra forma, el servidor es la parte intermedia entre los dos módulos, siendo el distribuidor entre ambos, sea para recibir como para responder a la solicitud.

Debido a que la práctica está comentada en su totalidad en **JavaDoc**, se documentan aun así los servicios y métodos más significativos de la práctica para explicar su funcionamiento y tomas de decisión. Se asume que cada proyecto tiene publicados correctamente cada servicio.

Callback

El usuario crea un objeto remoto el cual tiene acceso a escribir en su propia consola, siendo este objeto el que se envía hacia la base de datos a través del servidor en la fase de **login** como parámetro, y el que utilizará la base de datos para notificarle información al usuario.

Se ha tomado esta decisión, porque es mejor que el uso de un “buzón” y así poder liberar al usuario de consultas sobre el objeto para verificar si hay elementos nuevos. De esta otra forma es el propio **callback** el que notifica al usuario, pasando el usuario de ser un elemento activo a pasivo.

Login de acceso

A nivel del proyecto de **usuario** se define como constante el tipo de servicio, puerto y URL donde hará la llamada para hablar con el módulo del servidor:

```
public class MiniTwitter_Usuario {

// variables para conectar a los servicios del servidor
final static int port = 5556;
final static String URL_Autentificador = "rmi://localhost:" + port + "/Autentificador/Autentificador";
final String URL_Gestor = "rmi://localhost:" + port + "/Gestor/Gestor";

private ServicioAutentificacionInterface logueador;
private ServicioGestorInterface gestor;

[…]

public void iniciar() throws RemoteException, NotBoundException {
	Registry registro = LocateRegistry.getRegistry("localhost", port);
	logueador = (ServicioAutentificacionInterface)   registro.lookup(URL_Autentificador);
```

Cuando elige la opción de logar, se llama desde el servicio `logueador` (Servicio **Autenticación**) al método **login** del servidor donde informa de los campos alias/Nick y password introducidos previamente por teclado, yendo la petición contra el servidor y comprobando éste en la base de datos si los campos son correctos, permitiendo así el acceso a `gestor` (Servicio **Gestor**).

```
[…]

case 2:
	System.out.print(">> Escriba su login: ");
	String login = teclado.nextLine();
	System.out.print(">> Escriba el password de @" + login + ": ");
	String pass = teclado.nextLine();
	if (logueador.login(login, pass, retorno)) {
		gestor = (ServicioGestorInterface) registro.lookup(URL_Gestor);
		this.alias = login;

	}
[…]
```

Mientras tanto el módulo **servidor** tiene publicado sus servicios (**Autenticador** y **Gestor**), y además el acceso al módulo de base de datos, `bbdd`, vía servicio **Datos**

```
public class Servidor {

// variables usadas para la llamada a sus servicios así como para mostrar la información
por pantalla (info servidor)
private ServicioDatosInterface bbdd;
final int portEntrada = 5555;
final int portSalida = 5556;
final String URL_BBDD = "rmi://localhost:" + portEntrada + "/BBDD/BBDD";
final    String    URL_Autentificador    =    "rmi://localhost:"    +    (portSalida)    +
"/Autentificador/Autentificador";
final String URL_Gestor = "rmi://localhost:" + (portSalida) + "/Gestor/Gestor";
ServicioAutentificacionInterface autentificador;
```

```
[…]
/**
* constructor de la clase servidor
*/
public Servidor() {
	try {
		Registry registro = arrancarRegistro(portEntrada);
		ServicioGestorImpl gestor = new ServicioGestorImpl(this);
		autentificador = new ServicioAutentificacionImp(this, gestor);

		Registry registro2 = arrancarRegistro(portSalida);
		registro2.rebind(URL_Gestor, UnicastRemoteObject.exportObject(gestor, portSalida));
		registro2.rebind(URL_Autentificador,   UnicastRemoteObject.exportObject(autentificador,
portSalida));
		bbdd = (ServicioDatosInterface) registro.lookup(URL_BBDD);
[…]
```

El servidor recibe la petición del usuario para logar y éste se la envía al módulo de base de datos:

```
boolean login(String alias, String pass, CallbackUsuarioInterface callback) throws RemoteException {
	try { return bbdd.autenticar(alias, pass, callback);}
	catch (RemoteException ex) {System.out.println("\n[FAILED] Error en el login del usuario " +
alias);
```

Por último, el módulo de **base de datos** publica su servicio, **Datos**, esperando a que le lleguen las peticiones del servidor para gestionarlas. Cuando el usuario se loguea guarda su callBack para poder comunicarse con él.

En este punto si los usuarios a los que sigue han publicado/enviado trinos éstos se le mostrarán al usuario una vez logue al sistema a través del callBack.

```
final static int PUERTO = 5555;
final static String URL_BBDD = "rmi://localhost:" + PUERTO + "/BBDD/BBDD";

  […]
public static void main(String[] args) {
	Registry registro = null;
	try {
		registro = LocateRegistry.createRegistry(PUERTO);
		Basededatos BBDD = new Basededatos();
		ServicioDatosImpl servicioDatos = new ServicioDatosImpl(BBDD);
		registro.rebind(URL_BBDD, (ServicioDatosInterface)
UnicastRemoteObject.exportObject(servicioDatos, PUERTO));
		initBBDD();
		 while (true) {
		     BBDD.menu();
		}

  […]
```

```
[…]

public boolean autenticar(String alias, String password, CallbackUsuarioInterface
notificador) {
        if (!usuarios.containsKey(alias))
            return false;

        boolean correcto = usuarios.get(alias).chequeaPassword(password);
        if (correcto) {
            usuarioOnline.put(alias, notificador);
            List<TrinoInterface> pendientes = trinosPendientes.get(alias);
            for (TrinoInterface trino : pendientes) {
                try {  notificador.mandar(trino); }
                catch (RemoteException ex) {
  […]
```

Registro de usuario

Para éste y los siguientes casos no se vuelve a explicar la publicación de los servicios, ya que se ha informado en el punto anterior. El usuario solicita hacer el alta:

```
[…]
            case 1:
                altaUsuario();
                System.out.print("\n\n>> Pulse intro para continuar... ");
                intro = pedirIntro();
                break;

[…]
```

Llama al servicio de **Autenticación** (propio del servidor).

```
[…]

private void altaUsuario() throws RemoteException, NotBoundException {
        Scanner teclado = new Scanner(System.in);
        System.out.print("> Escriba su login de acceso: ");
        String login = teclado.nextLine();
        System.out.print("> Escriba su nombre: ");
        String nombre = teclado.nextLine();
        System.out.print("> Escriba su password: ");
        String pass = teclado.nextLine();
        if (logueador.registro(login, nombre, pass)) {
            System.out.println("\n[OK] Alta realizada correctamente");
            System.out.print("\n\n>> Pulse intro para continuar... ");
            String intro = pedirIntro();
        } else {
[…]

public boolean registro(String alias, String nombre, String pass) throws
RemoteException {
      return servidor.registrarUsuario(new Usuario(alias, nombre, pass));
}

[…]

public boolean registrarUsuario(UsuarioInterface usuario) throws RemoteException {
        return bbdd.addNuevoUsuario(usuario);
}
```

Cuando está en la base de datos se comprueba que dicho usuario no existe y en este caso se crean las estructuras de datos que necesite, como su colección de trinos, seguidores….

```
[…]

// coleccion de datos
private HashMap<String, UsuarioInterface> usuarios;
private HashMap<String, LinkedList<TrinoInterface>> trinos;
private HashMap<String, ArrayList<UsuarioInterface>> seguidores;
private HashMap<String, CallbackUsuarioInterface> usuarioOnline;
private HashMap<String, ArrayList<TrinoInterface>> trinosPendientes;

[…]

public boolean addNuevoUsuario(UsuarioInterface usuario) {
        if (usuarios.containsKey(usuario.getLogin()))
            return false;

        usuarios.put(usuario.getLogin(), usuario);
        trinos.put(usuario.getLogin(), new LinkedList<TrinoInterface>()); // Linked
debido al tipo de acceso y los borrados
        seguidores.put(usuario.getLogin(), new ArrayList<UsuarioInterface>());
        trinosPendientes.put(usuario.getLogin(), new ArrayList<>());
        return true;
}

[…]
```

Seguir a un usuario

El módulo usuario realizará los mismos pasos que cuando se loga, pero en el momento de querer seguir a alguien hará uso del servicio **Gestor**, en vez de **Autenticar** como en los casos anteriores.

```
[…]

private void seguirUsuario() throws RemoteException {
        System.out.print(">> Escriba el alias del usuario que desea seguir: ");
        String seguir = new Scanner(System.in).nextLine();
        if (gestor.seguirA(alias, seguir)) {
            System.out.println("\n[OK] Ahora está siguiendo a... " + seguir);
        } else {
            System.out.println("\n[KO] Imposible seguir a... " + seguir);
        }

[…]
```

La petición llega al servidor y se la proporciona a la base de datos vía servicio **Datos** para que la gestione.

```
[…]

boolean seguirA(String quienSoy, String aQuienQuieroSeguir) throws RemoteException
{
        try {
            return bbdd.follow(quienSoy, aQuienQuieroSeguir);
        } catch (RemoteException ex) {
[…]
```

Y la base de datos añadirá, si todo es correcto, el usuario en la colección de **seguidores** devolviendo a servidor y éste al usuario la respuesta a la petición original.

```
[…]

public boolean follow(String quienSoy, String aQuienQuieroSeguir) {
        if (seguidores.containsKey(aQuienQuieroSeguir)) {
            UsuarioInterface usserSoy = usuarios.get(quienSoy);
            UsuarioInterface usserSigo = usuarios.get(aQuienQuieroSeguir);
            List<UsuarioInterface>              seguidoresActuales              =
seguidores.get(aQuienQuieroSeguir);

            if          (!quienSoy.equalsIgnoreCase(aQuienQuieroSeguir)          &&
!seguidoresActuales.contains(usserSoy)) {
                seguidoresActuales.add(usserSoy);
                usserSoy.seguirA(usserSigo);

                return true;
            }
        }

        return false;
    }

[…]
```

Dejar de seguir a un usuario

El proceso es idéntico al de **Seguir a un usuario** (usuario ←→ servidor ←→base de datos) con la excepción que el usuario informado es eliminado de la colección de seguidores de la base de datos.

```
[…]

public boolean dejarDeSeguir(String aQuienNoQuieroSeguir, String quienSoy) throws
RemoteException {
        UsuarioInterface actual = usuarios.get(quienSoy);
        if (seguidores.containsKey(aQuienNoQuieroSeguir)) {
            return seguidores.get(aQuienNoQuieroSeguir).remove(actual) &
actual.dejarSeguir(usuarios.get(aQuienNoQuieroSeguir));
        }

        return false;
    }

[…]
```

Enviar trino

El módulo usuario se loga realizando la petición al servicio **Autenticación** y tras ello hará uso del servicio **Gestor** para enviar los trinos.

```
[…]
private void enviarTrino() throws RemoteException {
       System.out.print(alias + ">> ");
       String mensaje = new Scanner(System.in).nextLine();

         gestor.enviarTrino(new Trino(mensaje, alias));
    }
[…]
```

Y servidor se lo proporcionará a la base de datos vía servicio **Datos**.

```
public void enviarTrino(TrinoInterface trino) throws RemoteException {
        servidor.addTrino(trino);
}

[…]

public void addTrino(TrinoInterface trino) throws RemoteException {
        bbdd.addTrino(trino);
}

[…]
```

Añadiendo el respectivo trino en la colección de la base de datos.

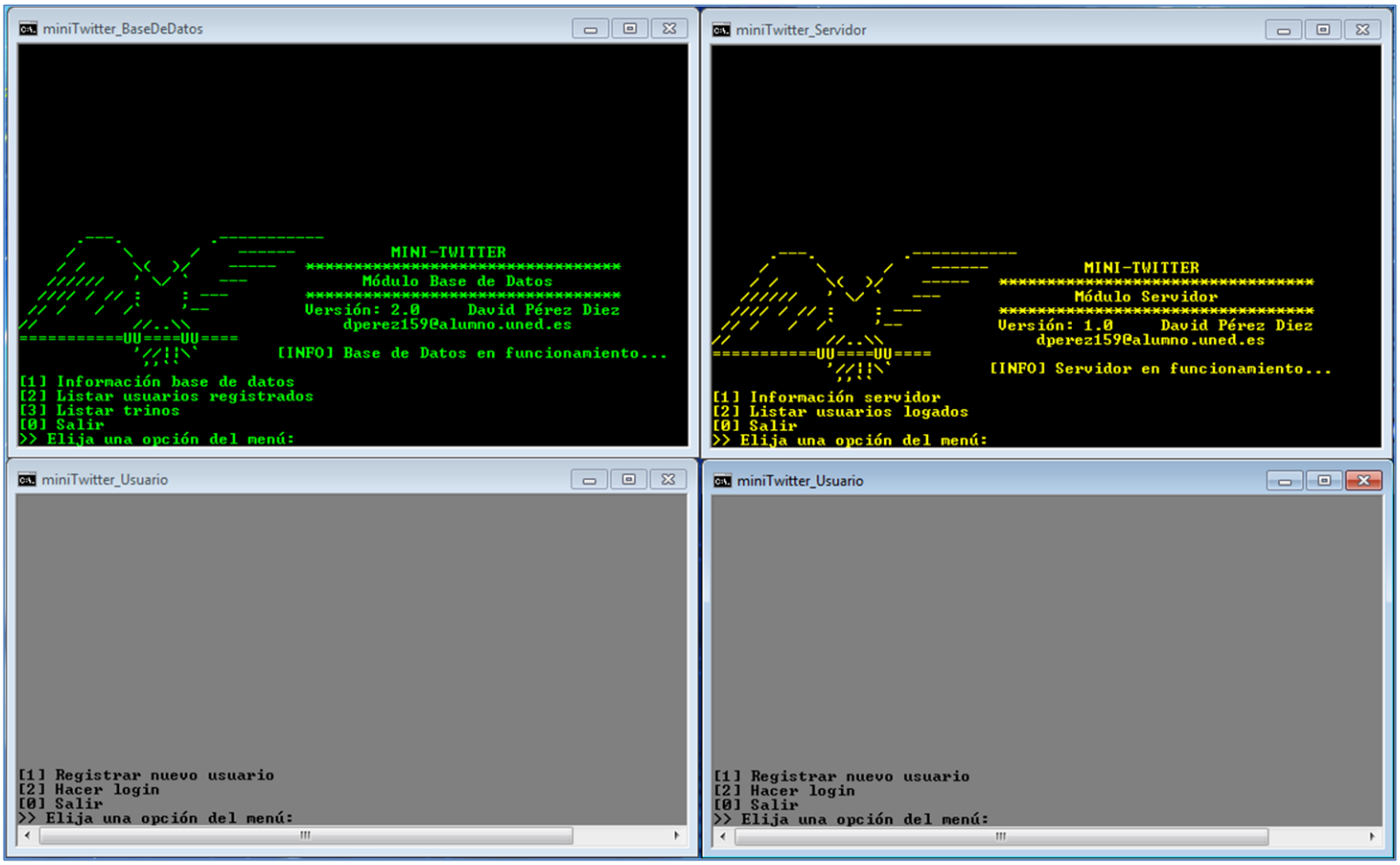
```
[…]

public void addTrino(TrinoInterface trino) throws RemoteException {
        Trino nuevo = new Trino(trino.ObtenerTrino(),
trino.ObtenerNickPropietario()); // guarda los trinos enviados de los usuarios
        trinos.get(trino.ObtenerNickPropietario()).add(nuevo);
        // se manda a los que siguen al usuario
        for (UsuarioInterface seguidor :
seguidores.get(nuevo.ObtenerNickPropietario())) {

            if (!usuarioOnline.containsKey(seguidor.getLogin())) {
                trinosPendientes.get(seguidor.getLogin()).add(nuevo);
            } else {
                try {
                    usuarioOnline.get(seguidor.getLogin()).mandar(nuevo);
                } catch (RemoteException ex) {
                    System.out.println("\n[FAILED] Error al notificar mensaje al
usuario @" + seguidor.getLogin());
                }
            }
        }
}

[…]
```

➢ FUNCIONAMIENTO DEL SISTEMA

A continuación se describe mediante un ejemplo el funcionamiento del sistema en **ausencia de errores** con la finalidad de no extender demasiado su explicación.

A nivel informativo, hay un video publicado en **YOUTUBE** con una batería de pruebas más exhaustiva.

Para iniciar el sistema es necesario ejecutar en orden los respectivos binarios como se ha explicado en el en el punto **Uso de la práctica**, primero `miniTwitter_BaseDeDatos.cmd`, seguido de `miniTwitter_Servidor.cmd` y por último `miniTwitter_Usuario.cmd`, o bien ejecutar el script para tal fin, `arranque_controlado.cmd`, ya que se recomienda arrancar dos clientes/usuarios para comprobar cómo interactúan entre ellos.



lustración 6 - Módulos de la práctica

Una vez arrancados los módulos se detallan las secuencias de las acciones que el usuario final debe realizar, y el impacto que tienen en el resto de los módulos, base de datos y servidor.

Interface de Usuario

Registrar usuario

Se hace a través de la propia interface de usuario, previamente pulsando la **opción 1** una vez arrancada y visible.

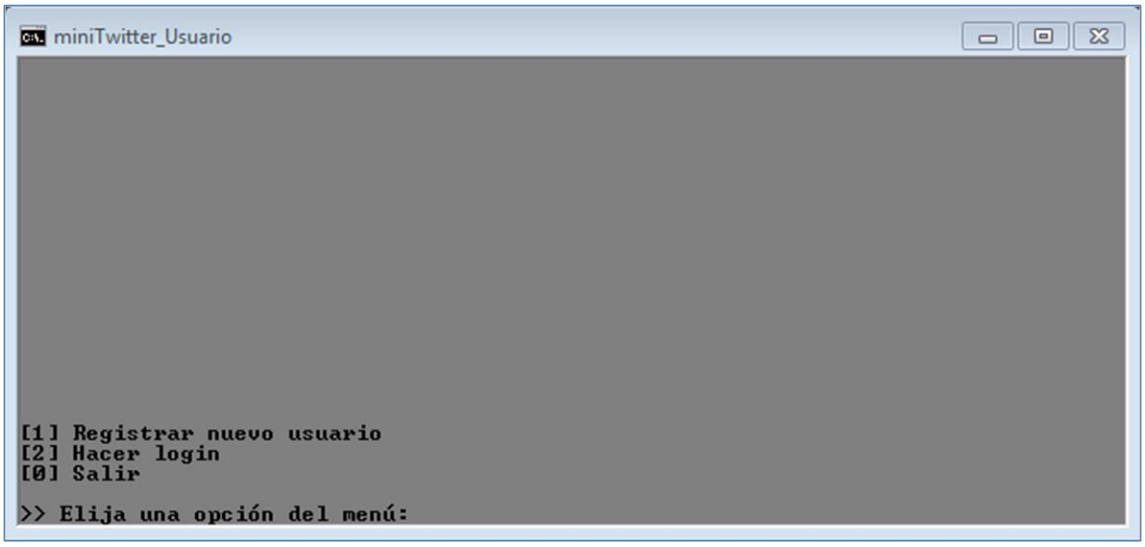


Ilustración 7 - Menú de acceso

Llevado a cabo el registro se informará al usuario del proceso vía consola. Se procede a crear dos registros para que puedan interactuar entre ellos.

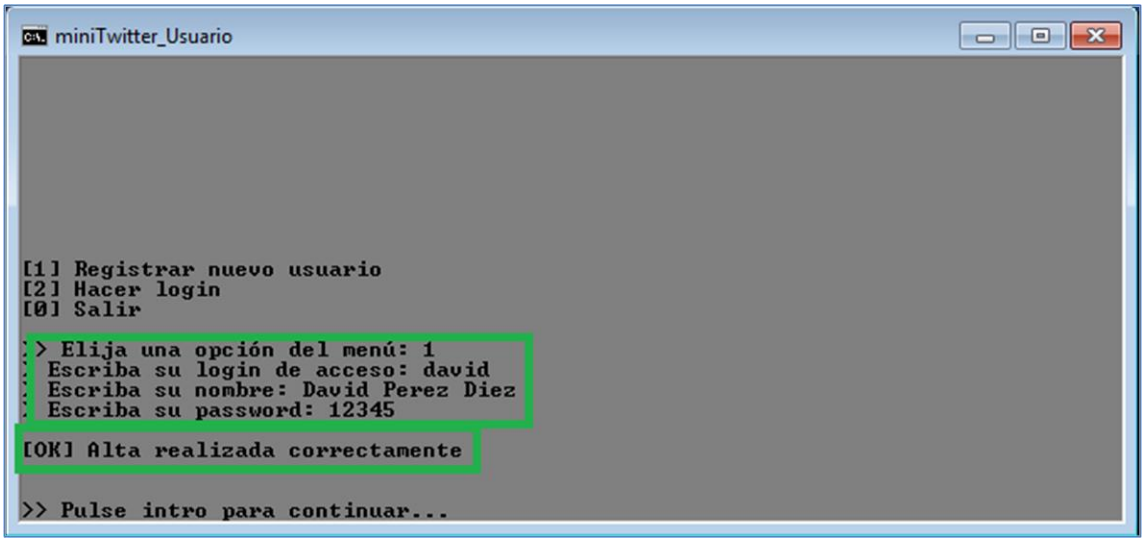


Ilustración 8 - Registro de usuario 1

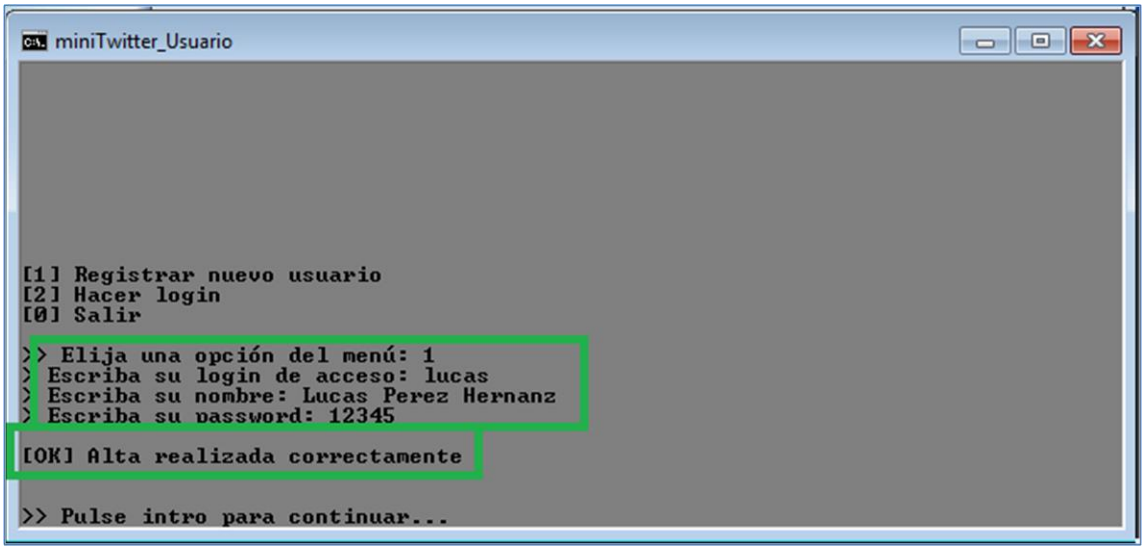


Ilustración 9 - Registro de usuario 2

Iniciar sesión

Se realiza sobre la misma interface, pero esta vez pulsando la **opción 2** e indicando las credenciales informados en el punto anterior.

Si la validación es correcta se mostrará un nuevo menú con las opciones para interactuar con las funcionalidades que proporciona el sistema.

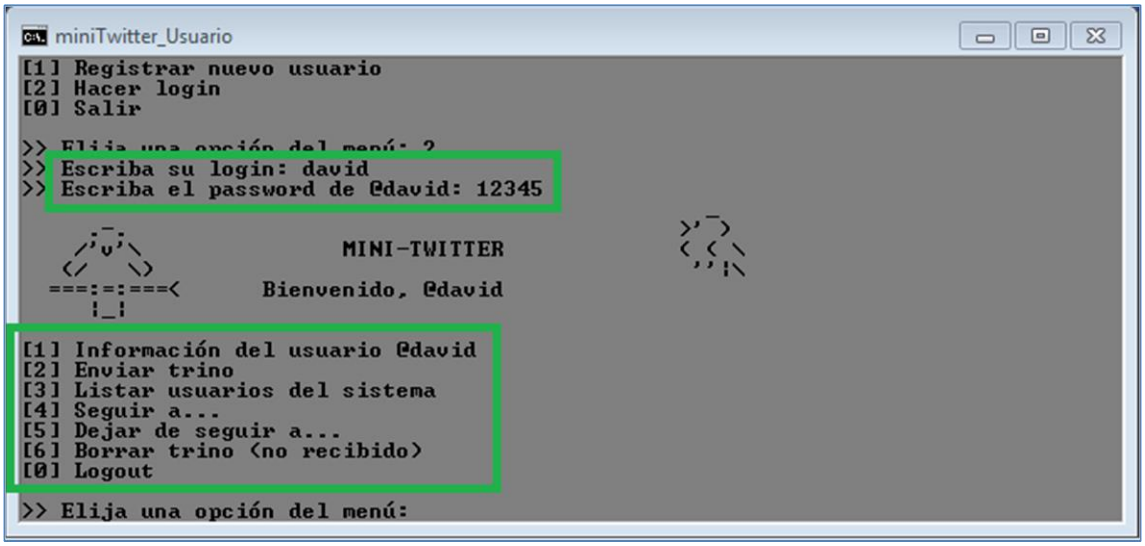


Ilustración 10 - Login de usuario

Información del usuario

El usuario arranca un servicio, que es el propio `Callback`, el cual de forma dinámica habilita un puerto, pero no se enlaza a ningún registro, por lo que se ha tomado la decisión de mostrar información relevante a nivel de los servicios que consume el usuario, como son el **Servicio Autenticar** y el **Servicio Gestor**.

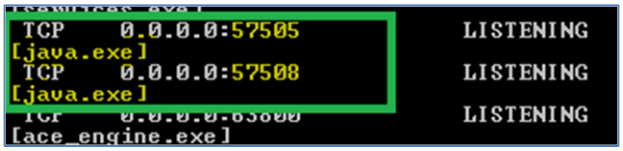


Ilustración 11 - Puertos dinámicos habilitados por cada ejecución de usuario

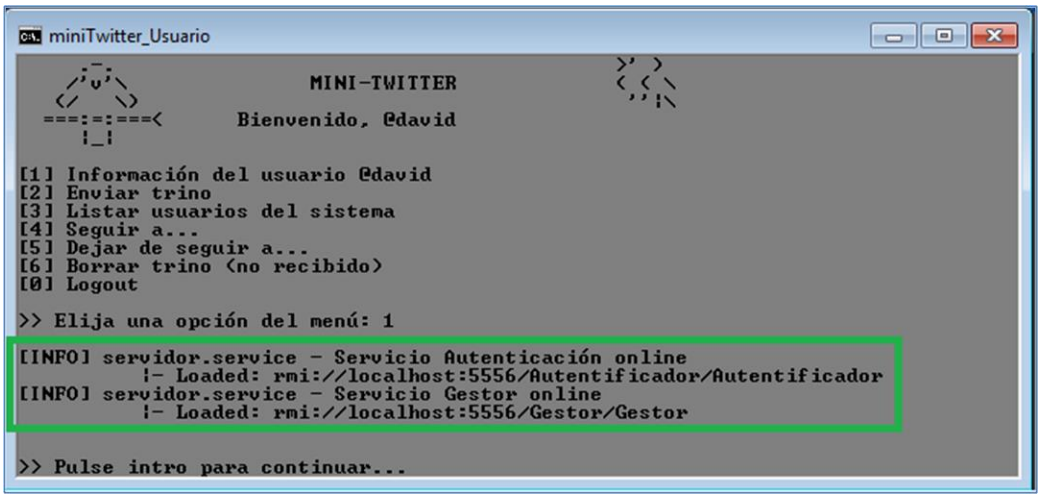


Ilustración 12 - Información usuario

Seguir a

Con esta acción un usuario podrá seguir a cualquier otro usuario que se encuentre dado de alta en el sistema, por lo que es necesario tener al menos dos usuarios registrados. Ver **Registrar usuario** e **Iniciar sesión**.

Para facilitar esta acción en **Listar usuarios del sistema** se ofrece una lista de qué usuarios hay actualmente registrados y a cuáles se sigue por parte del usuario interactivo.

Una vez informado del alias del usuario que se quiere seguir, se mostrará un mensaje informativo de si dicha asignación ha sido correcta o no (en el caso que el usuario no exista o esa él mismo).

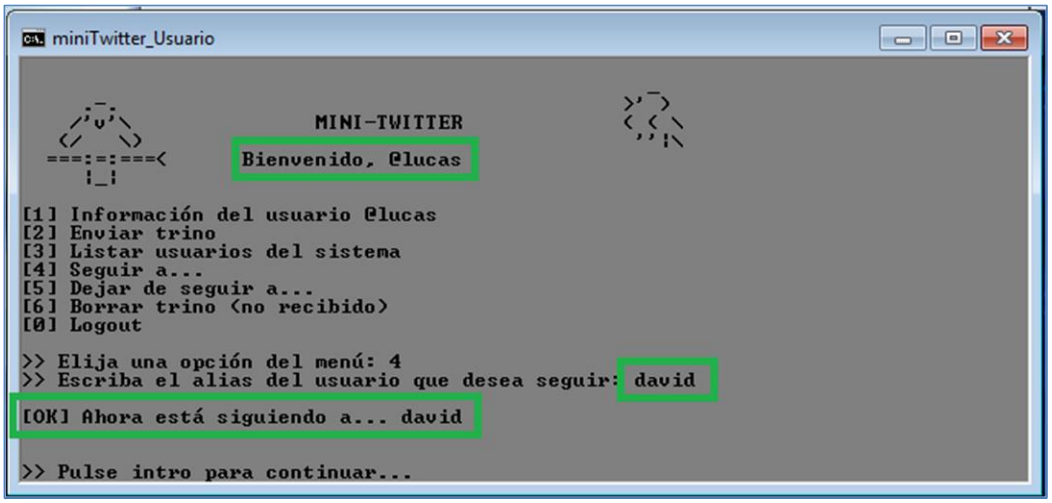


Ilustración 13 - Seguir a un usuario

Listar usuarios del sistema

Muestra los usuarios registrados en el sistema más aquellos de los cuales sigue, para así facilitar la acción de **Seguir a**.

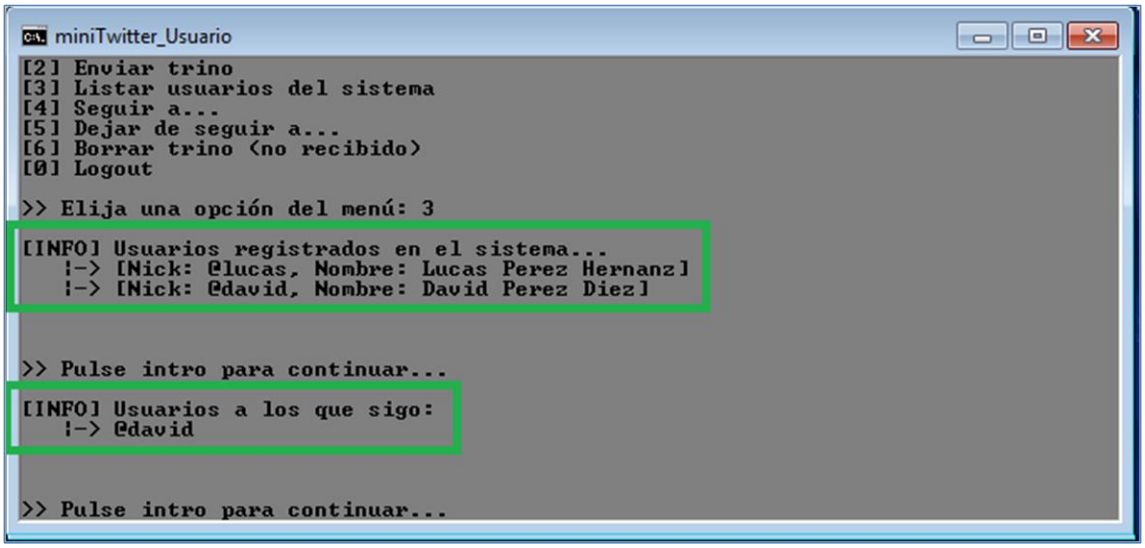


Ilustración 14 - Listado de usuarios y a quién se sigue

Enviar trino

Esta operación enviará un trino a todos los seguidores que se tengan en ese momento, de tal manera que aquellos que se encuentren logados recibirán el trino instantáneamente, mientras que los desconectados los recibirán cuando accedan al sistema, excepto aquellos que se hayan borrado previamente, tarea opcional (**Borrar trino**).

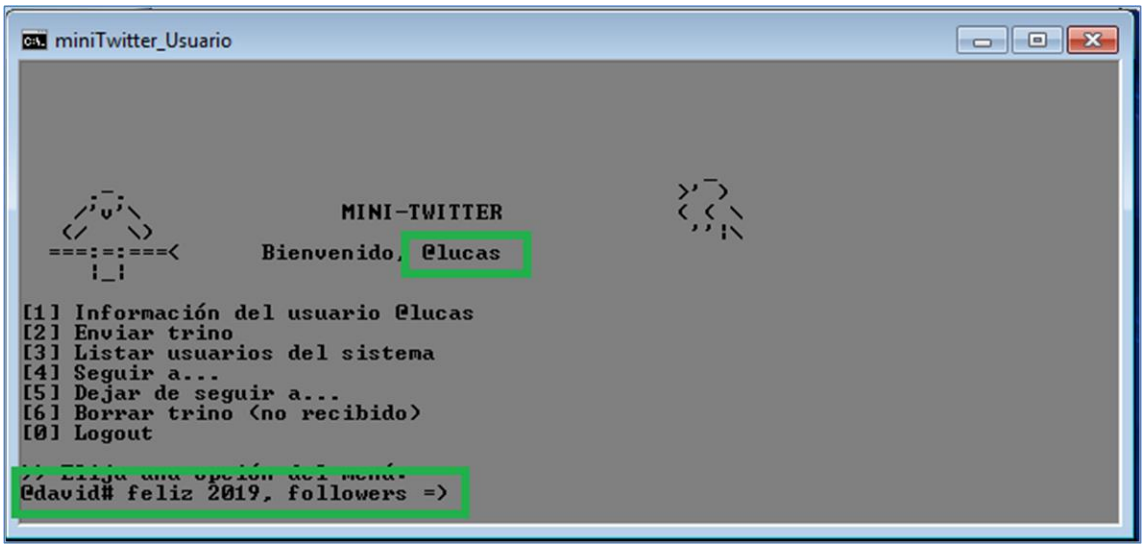


Ilustración 15 – Envío de un trino a los seguidores logados

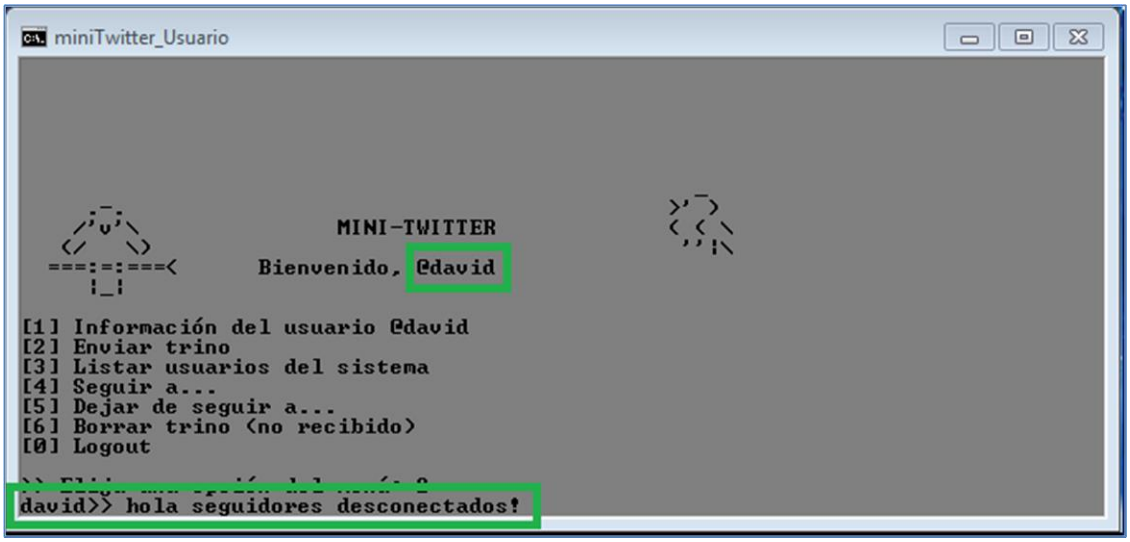


Ilustración 16 - Envío de un trino a los seguidores deslogados

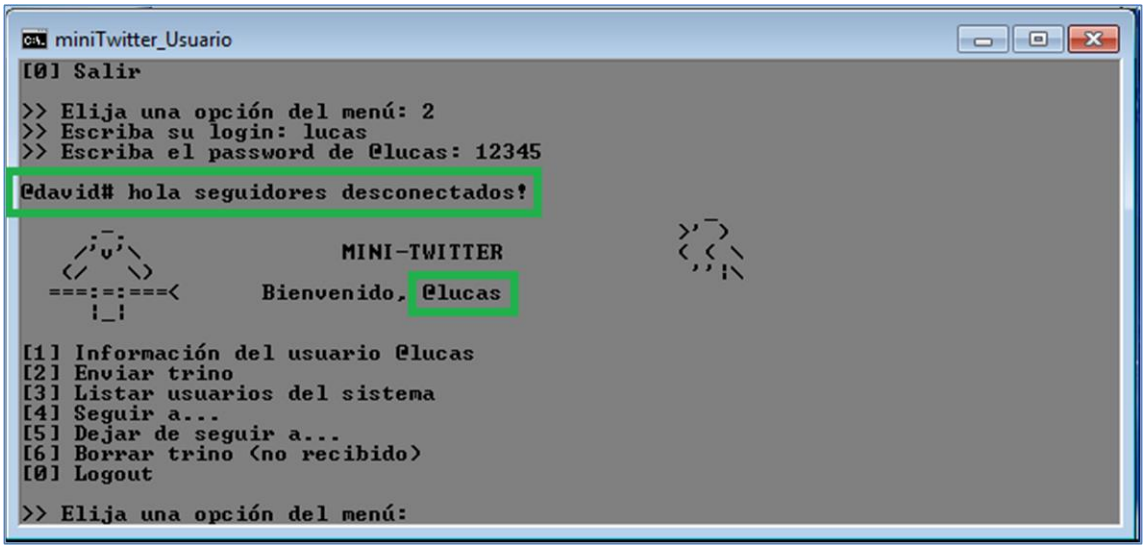


Ilustración 17 - Recepción de un trino al logar

A destacar que la acción de enviar trino se realiza con o sin seguidores, la diferencia es que el primero será recibido por sus respectivos seguidores, mientras que el último no lo recibirá nadie.

Borrar trino

Tarea opcional y relacionada con la posibilidad de enviar trinos a seguidores que no están logados en el sistema.

Cuando un usuario envía un trino a sus seguidores, aquellos que no estén logados no lo recibirán hasta que se loguen, como se ha explicado en el punto anterior, y en esta situación el usuario que ha enviado el trino tendrá la capacidad de borrarlo, y por lo tanto, aquellos seguidores que se loguen a posteriori no lo recibirán.

El borrado se trata como una pila, el ultimo enviado es el primero a ser eliminado.

Ilustración 18 - Envío de un trino que será borrado

Ilustración 19 - Solicitud de borrar el último trino publicado

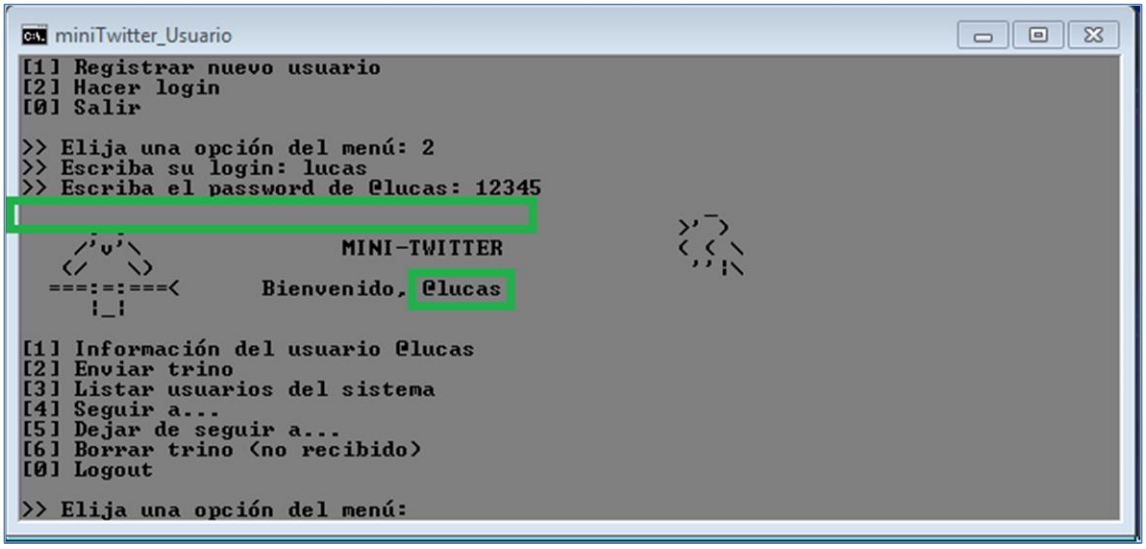


Ilustración 20 - Trino no publicado en el seguidor

Dejar de seguir

Con esta acción se deja de seguir a un usuario al que seguía con anterioridad. Se introduce el alias del usuario que se quiere dejar de seguir y mostrará un mensaje informativo de si dicha desasignación ha sido correcta o no (en el caso que el usuario no exista o no le siga).

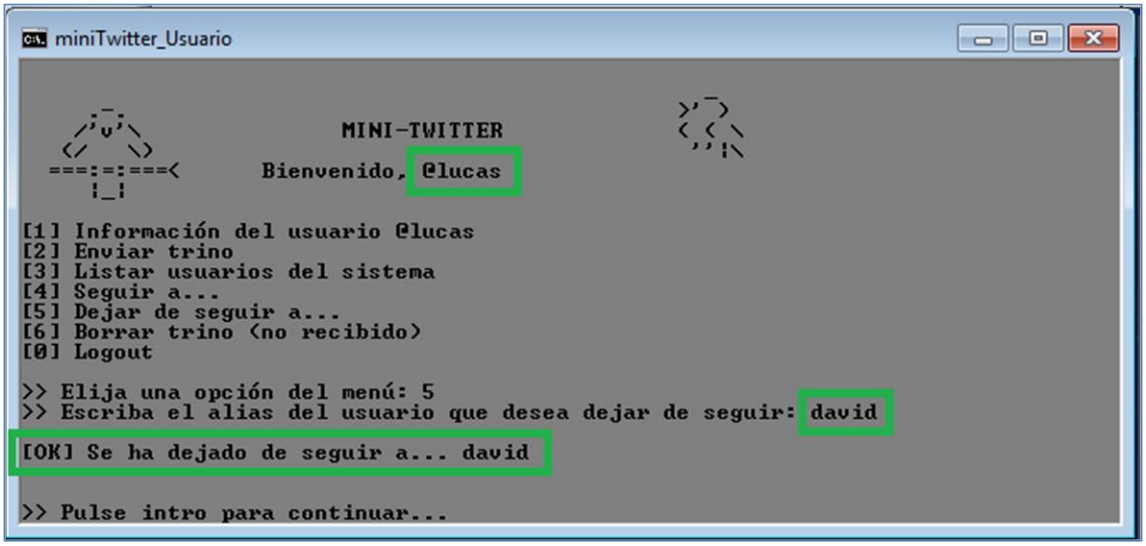


Ilustración 21 - Dejar de seguir a un usuario

Interface de Servidor

Una vez realizada las acciones informadas en la **Interface de Usuario**, creación y logueo de dos usuarios, se informa de la parte servidor.

Información del servidor

A diferencia del apartado **Información del usuario** aquí sí se registran los servicios, mostrando el detalle por pantalla.

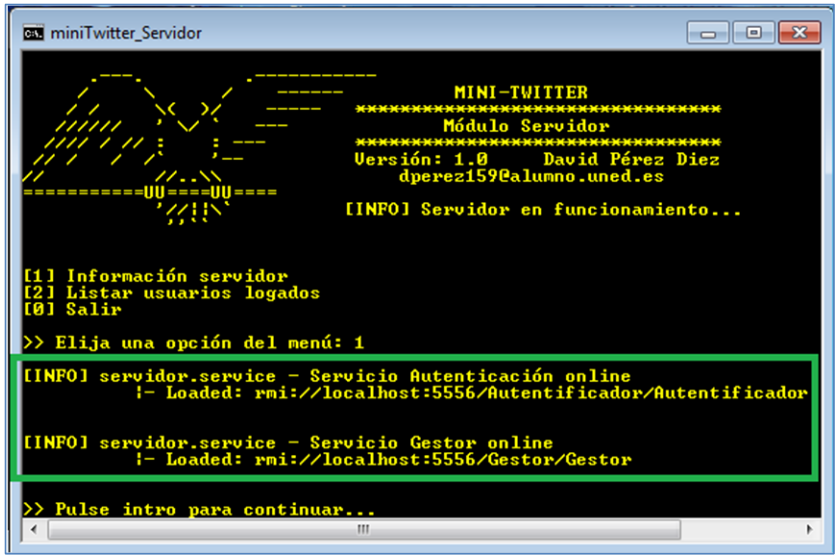


Ilustración 22 - Información del servidor

Listar usuarios logados

Muestra por pantalla los usuarios que han realizado la petición de iniciar sesión y han introducido correctamente las credenciales, indicando el alias de acceso junto con el nombre completo.

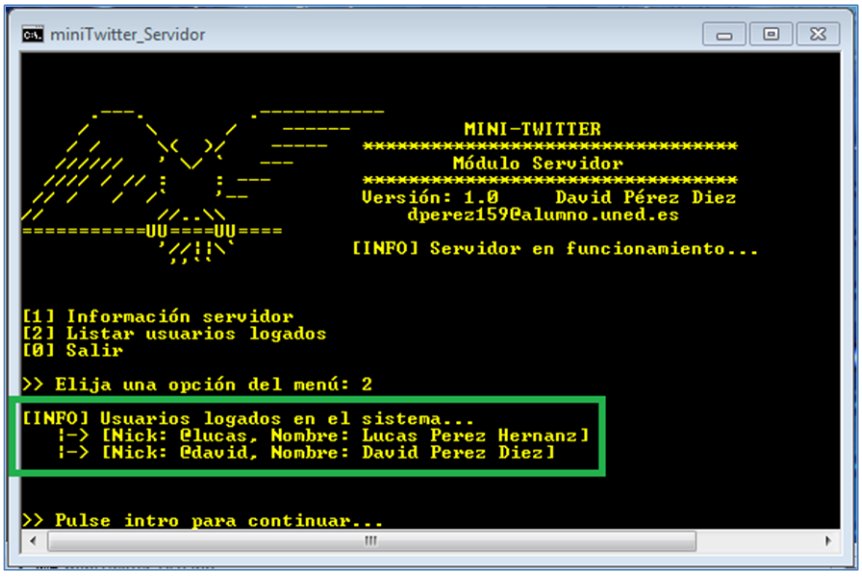


Ilustración 23 - Listado de usuarios logados en el sistema

Interface de Base de Datos

Información de la base de datos

Al igual que sucede en el servidor, registra en este caso un servicio, Servicio Datos, y lo muestra por pantalla.

En paralelo también muestra los usuarios registrados como logados (añadido a la práctica).

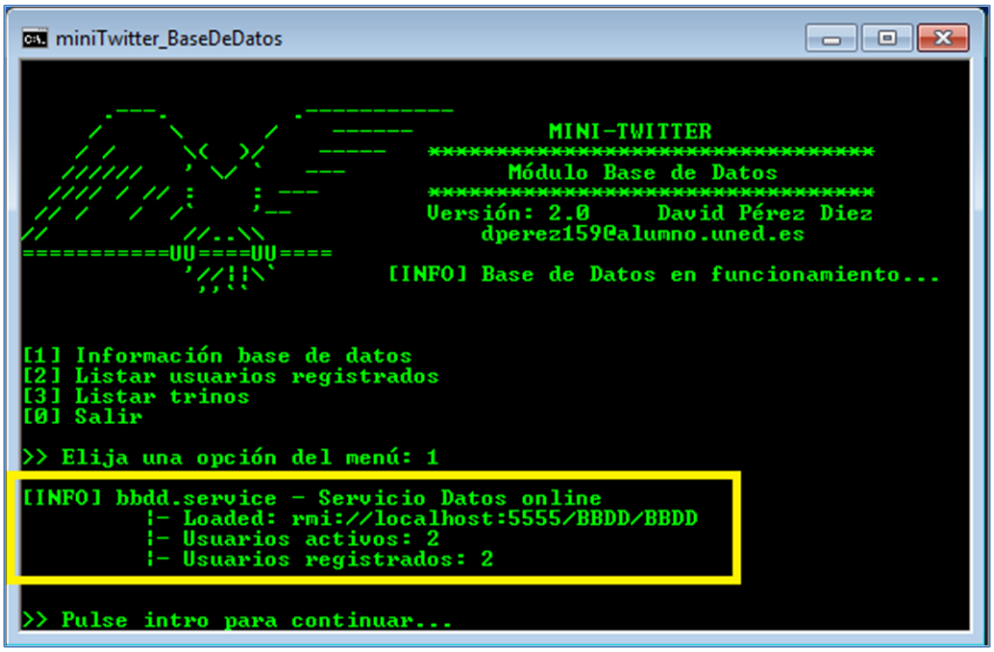


Ilustración 24 - Información de la base de datos

Usuarios registrados

Informa de los usuarios que se han creado/dado de alta en el sistema.

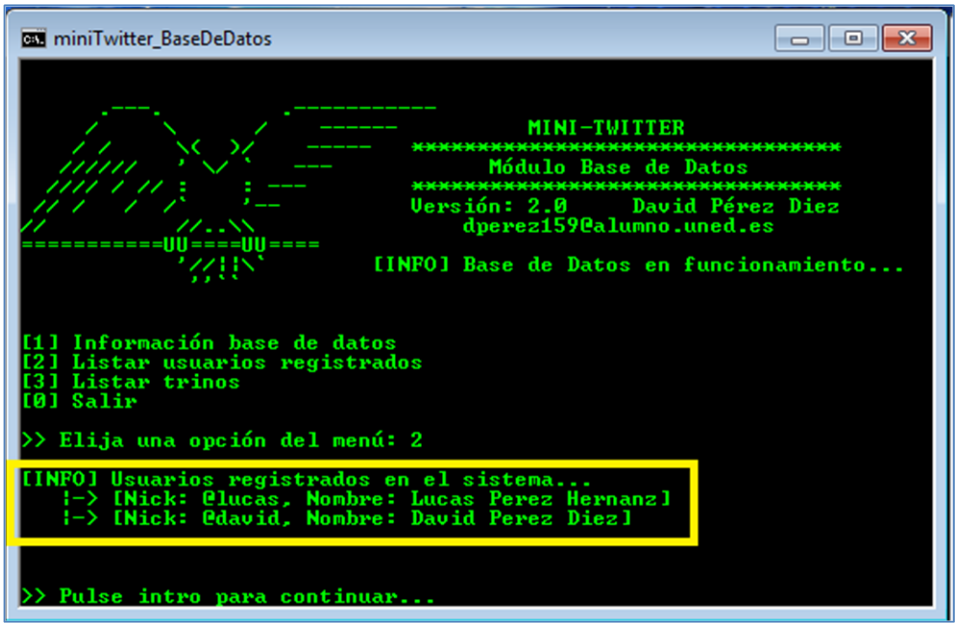


Ilustración 25 - Listado de usuarios registrados en el sistema

Listar trinos

Muestra por pantalla todos los trinos que se han enviado hasta el momento, se hayan o no recibido. A destacar que también se mostrarán aquellos que fueran borrados antes de que el usuario los recibiera, ya que la acción de envío es realizada satisfactoriamente, independientemente de la actividad del usuario (con o sin seguidores) o de su estado (logado o no).

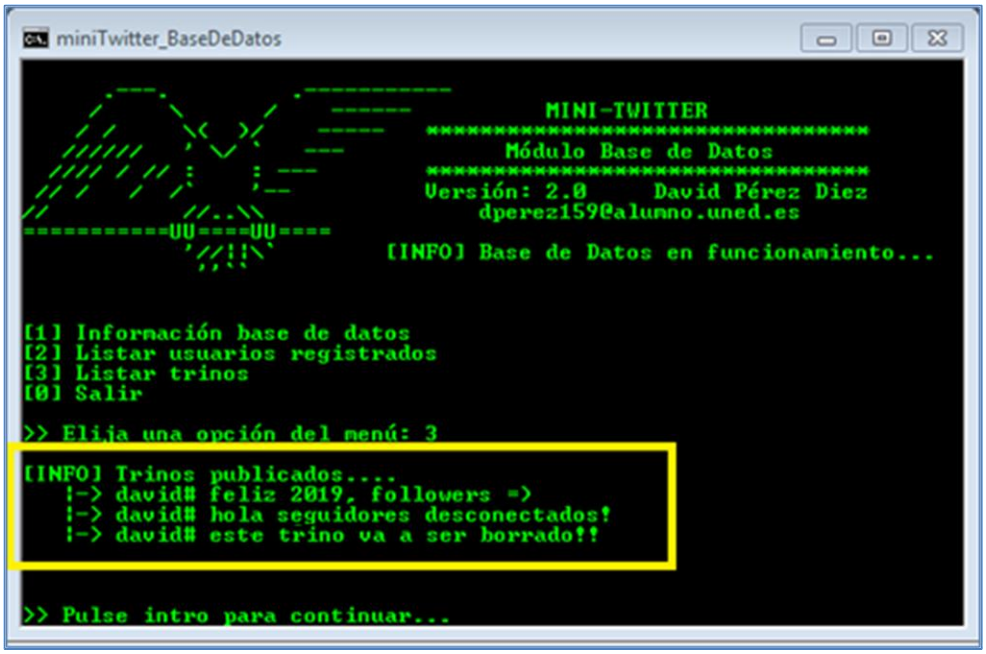


Ilustración 26 - Listado de trinos emitidos

➢ CONCLUSIONES Y MEJORAS

Tras la lectura de la memoria junto con su ejemplo y vídeo se verifica que el sistema alcanza los objetivos exigidos, incluyendo alguna funcionalidad no solicitada como ayuda al usuario que interactúa con la aplicación, más la tarea opcional de borrar trinos.

Con las limitaciones que proporciona una aplicación a nivel de consola o terminal en comparación con una interface gráfica se ha intentado dotar de un diseño amigable e intuitivo para poder facilitar la tarea al usuario.

Aun así, la práctica está sujeta a infinidad de **mejoras**, como por ejemplo:

- Diseñarlo en **modo gráfico**, de esta forma, además de ser más vistoso, permite la opción de imprimir los trinos recibidos en componentes de la propia ventana, como por ejemplo una lista de texto, y evitar que se solape con el menú de la interface.
- Mantener la **persistencia** de datos, haciendo uso por ejemplo de un SGBD.
- Añadir capas de **seguridad** entre las peticiones de los módulos, para que entre otras cosas los datos vayan cifrados.
- Mejor **control de las excepciones**, haciendo uso de `try-catch` en vez de `throws`.
- Ser más **estricto** con el MVC, ya que es la primera vez que hago uso de él.
- Control de **concurrencia** en el envío y recepción de trinos.

Como **opinión personal** me ha parecido una práctica interesante y novedosa al poder aplicar una arquitectura cliente-servidor, y más cuando nunca he tenido la posibilidad de trabajar con Java RMI, ayudándome en paralelo a comprender cómo y qué necesitan los clientes para comunicarse con los respectivos servicios, así como a la preparación del examen presencial de la asignatura.