Sistemas operativos en red

Sistema de compra y venta de arte mediante subastas

Víctor Manuel González Mira

74368280B

Contenido

[Diseño del sistema 2](#_Toc419024012)

[Interfaces de comunicación 8](#_Toc419024013)

[Colas y Topics 8](#_Toc419024014)

[Servicios Web 9](#_Toc419024015)

[Juddi 10](#_Toc419024016)

[Implementación del sistema 11](#_Toc419024017)

[Aplicación de escritorio Galería de Arte (.Net) 11](#_Toc419024018)

[Aplicación de escritorio Galería de Arte (Java) 15](#_Toc419024019)

[Aplicación de escritorio Museo (Java) 18](#_Toc419024020)

[Aplicación web (ASP) 19](#_Toc419024021)

[Despliegue y configuración 21](#_Toc419024022)

[Seguridad 24](#_Toc419024023)

[Pruebas 25](#_Toc419024024)

[Valoración del proyecto 26](#_Toc419024025)

# Diseño del sistema

La idea principal del proyecto nos propone crear un sistema de subastas donde galerías de arte, museos o usuarios particulares pudieran comprar y vender sus obras de arte. Para ello se establecen una serie de requisitos como son el uso de ActiveMQ para gestionar colas y topics, o Juddi para deslocalizar los servicios que ofrece el gestor de subastas, entre otros.

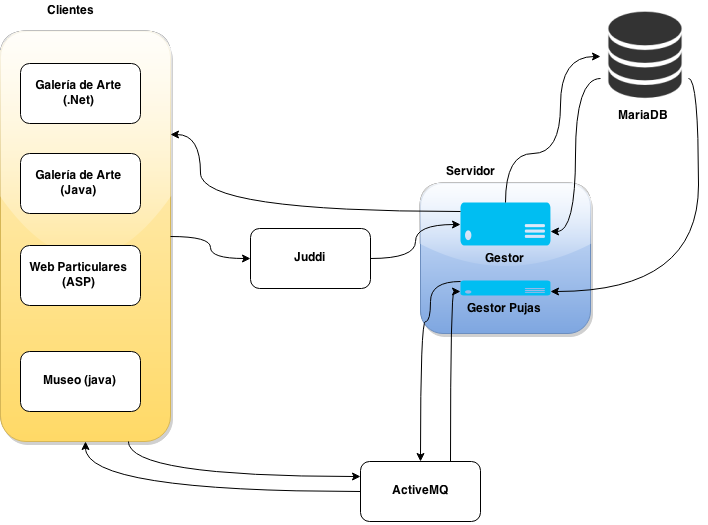
El resultado final es una serie de módulos independientes que dan forma al proyecto, quedando este estructurado de la siguiente manera:

* Gestor de subastas (WS)
  + Gestor de pujas
* Cliente de escritorio para galerías de arte (.NET)
* Cliente de escritorio para galerías de arte (Java)
* Cliente de escritorio para Museos (Java)
* Cliente web para usuarios particulares (.NET/ASP)

Todo ello apoyado por los siguientes servicios:

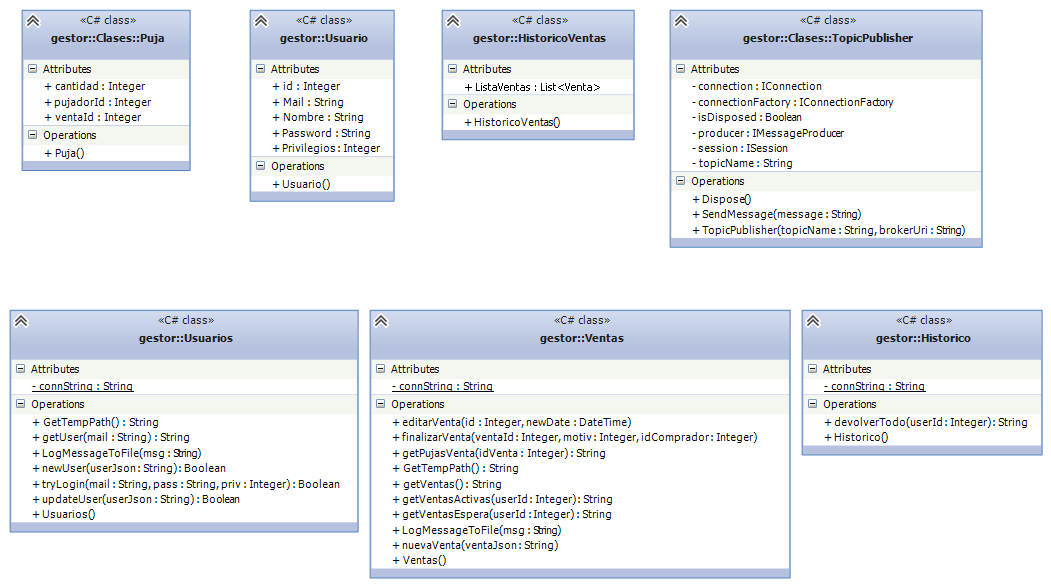
* ActiveMQ (Colas y Topics)
* Juddi (Registro de WS)
* MariaDB (Base de datos MySQL)

En el siguiente esquema conceptual veremos cómo se interrelacionan todos estos módulos para su funcionamiento.

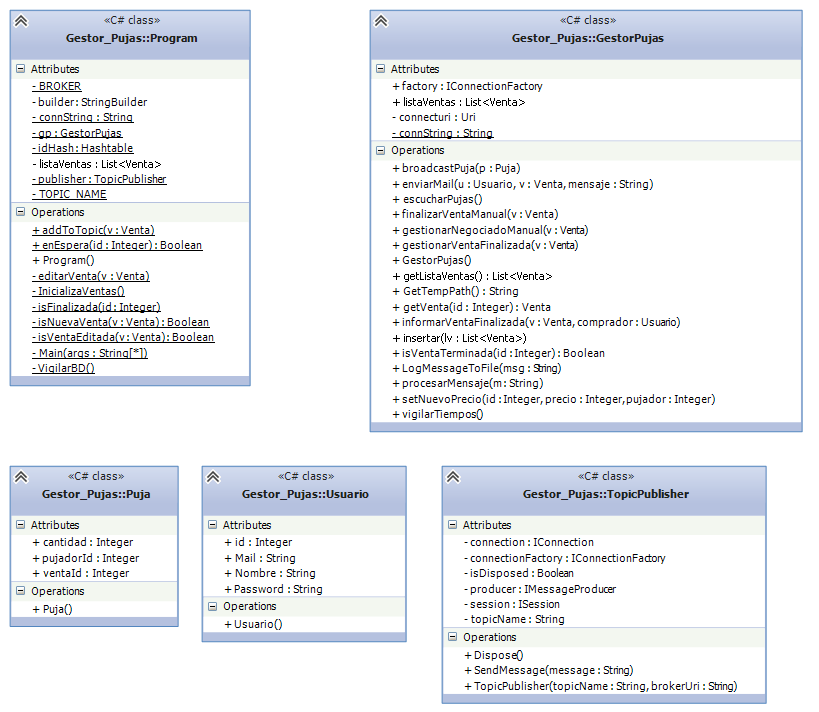


A continuación se muestran esquemas UML para cada uno de estos módulos.

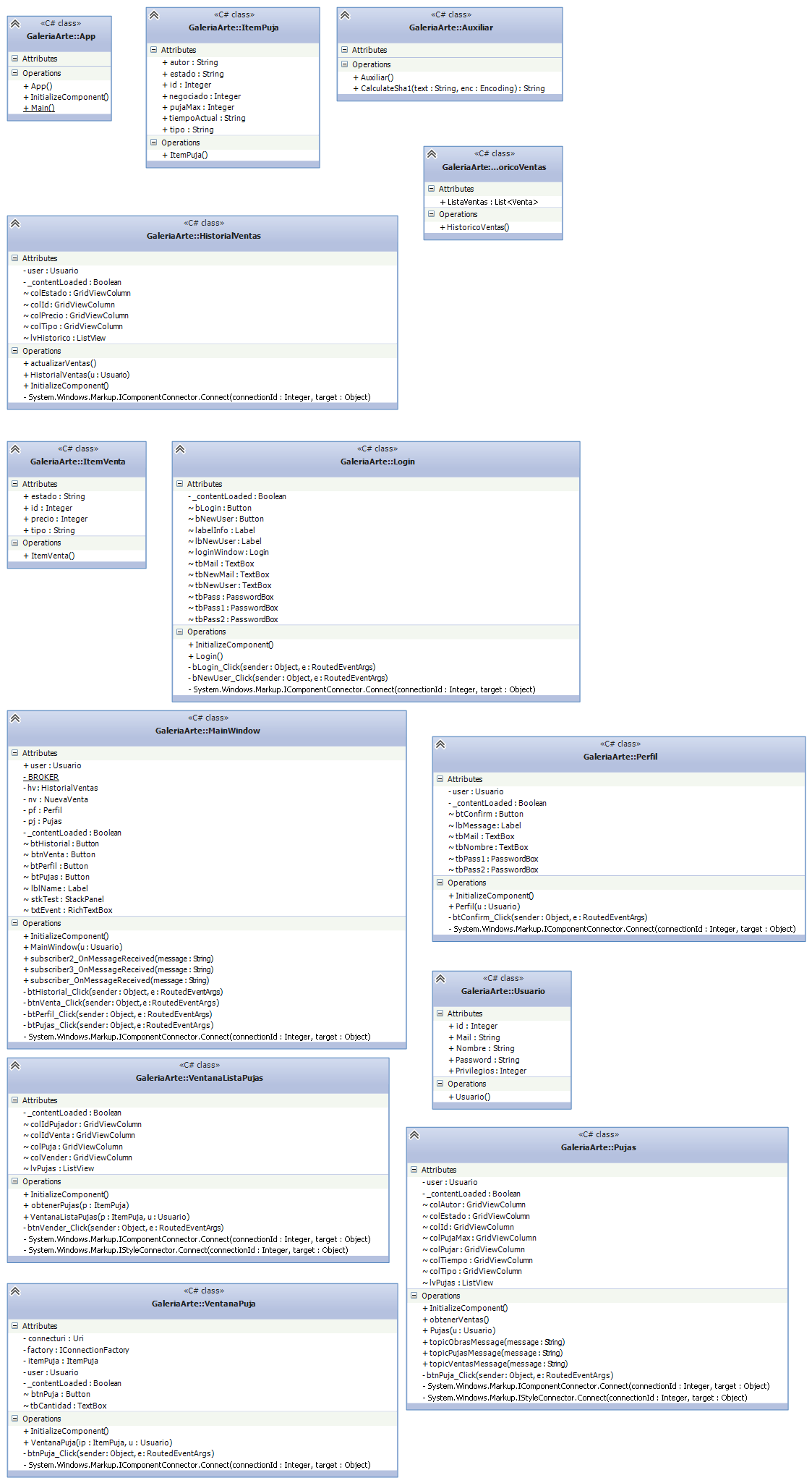
**Gestor**



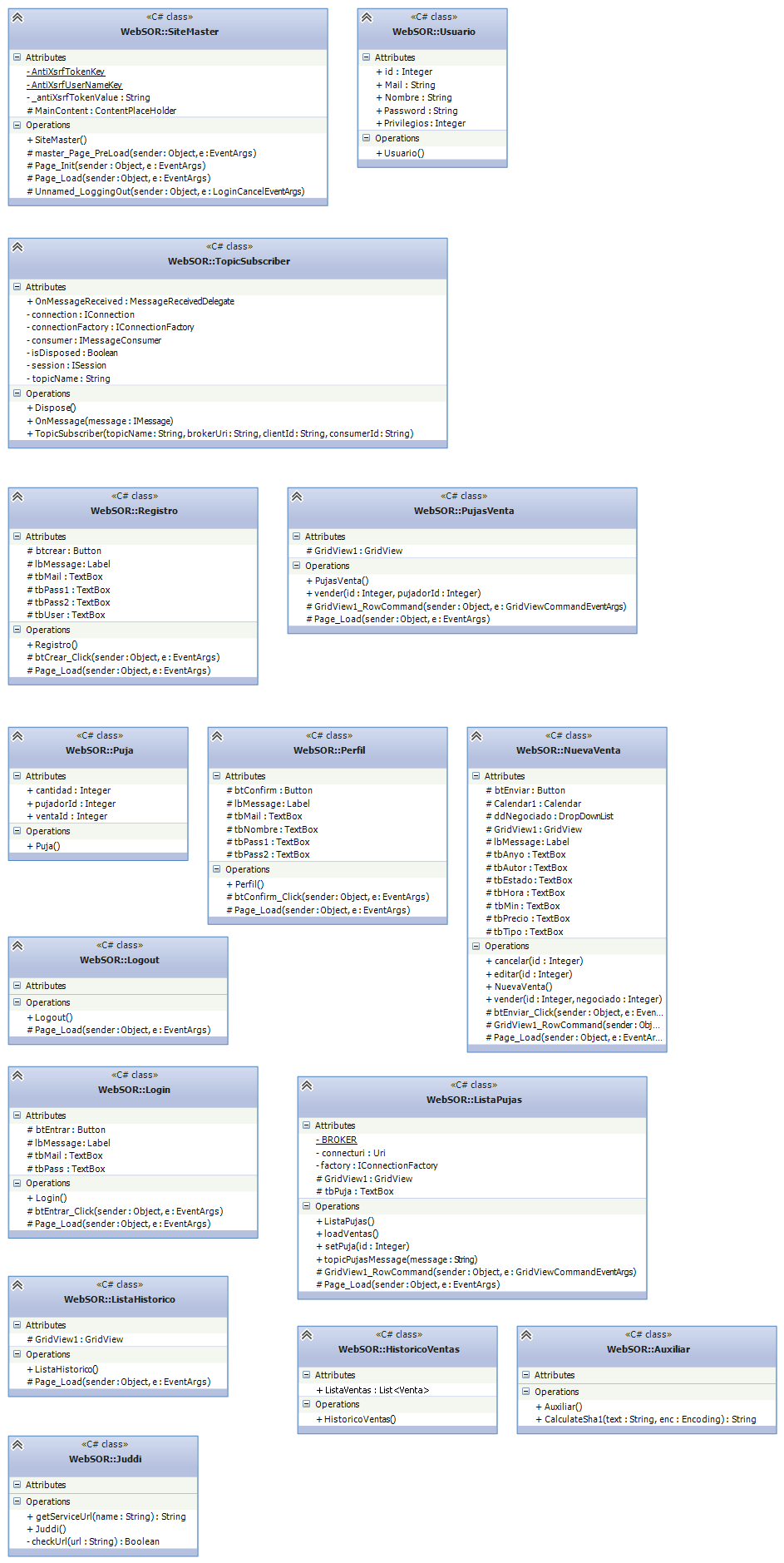
**Gestor Pujas**

****

**Galería de Arte**

****

**Web particular**

****

## Interfaces de comunicación

La comunicación entre las aplicaciones cliente y el gestor se realiza esencialmente en dos vertientes, ambas bidireccionales:

* Colas y topics
* Servicios web

### Colas y Topics

ActiveMQ nos proporciona lo que llamamos colas y topics, las cuales son en esencia, un sistema para el paso de mensajes en red. La diferencia entre colas y topics radica en lo siguiente:

* Colas: Un solo consumidor
* Topics: Múltiples consumidores

En nuestro caso las colas son útiles para gestionar el sistema de pujas ya que multiples clientes envían sus pujas a través de la cola para pujas y solo un cosumidor, el gestor, recoge esos mensajes.

En el caso de los topics, nos son útiles para emitir mensajes en broadcast desde el servidor que queremos que sean leídos por todos los clientes. Por ejemplo, la finalización y venta de alguna subasta o su cancelación.

ActiveMQ además nos permite hacer colas y topics de dos tipos durables y no durables cuya diferencia es que los durables permanecerán y serán cosumidos por los clientes cuando estos se conecten y los no durables solo serán entregados a los clientes que estén conectados en el momento de la emisión.

En nuestro caso nos interesa que la cola de pujas sea durable para que el gestor pueda recoger todas las pujas acumuladas durante una posible caída del sistema. En cambio los topics, al ser mensajes solo de interés para el momento específico en el que se envían, son no durables. No tendría sentido que un cliente se conectase y recibiese 40 mensajes de ventas concluidas en su ausencia.

A continuación se expone, por tanto, el uso que se hace de colas y topics en el proyecto:

* **Cola PUJAS**: Canal para el envió de pujas por parte de los clientes. Recepción de las mismas por parte del gestor de pujas.
* **Topic OBRAS**: Mensajes enviados por el gestor de pujas a los clientes con el objetivo de informar de la puesta en venta de una nueva obra de arte.
* **Topic PUJAS**: Mensajes enviados por parte del gestor de pujas a los clientes con el objetivo de informar de una nueva puja en alguna de las obras a la venta.
* **Topic VENTAS**: Mensajes enviados por parte del gestor de pujas a los clientes con el objetivo de informar de la venta o cancelación de alguna obra de arte.

### Servicios Web

El otro canal de comunicaciones entre cliente clientes y servidor es a través de los servicios web que ofrece el gestor. Los servicios web permiten enviar y recibir datos del gestor. Esto es posible mediante la llamada a una serie de métodos públicos que expone el gestor y que son accesibles de forma remota. Los servicios web permiten el envío y recepción de cualquier tipo de dato como si de llamadas a métodos tradicionales se tratasen.

En este caso, se quería asegurar la máxima compatibilidad con los clientes Java y .NET. Por lo tanto se ha evitado el envío y recepción de objetos puros. En vez de ello, todo lo que reciben y todo lo que devuelven estos servicios web son cadenas de texto JSON.

El protocolo utilizado en todo caso para la comunicación mediante WS es el siguiente:

* Cliente: Creación del objeto a enviar, Usuario, Venta, etc.
* Cliente: Conversión del objeto a formato JSON. En .Net mediante las librerías propias y en java mediante la librería externa GSon.
* Cliente: Llamada al WS pasando como parámetro la cadena JSON.
* Gestor: Recepción de la cadena JSON.
* Gestor: Conversión de la cadena JSON a un objeto de la clase esperada.

Esto nos evita sobretodo posibles problemas de compatibilidad en el la recepción de objetos Java en el gestor desarrollado sobre .Net.

El uso de WS hace también que todo dato requerido por los clientes deba ser consultado directamente al gestor a través de un WS. Por lo tanto no existe el acceso directo de los clientes a la base de datos. Solo el gestor puede acceder a la misma.

### Juddi

Si hablamos de servicios web hemos de hablar también de Juddi.

Juddi es básicamente un registro de servicios web, una guía telefónica para saber dónde están localizados los WS y poder acceder a ellos aunque nosotros no sepamos directamente su dirección. Mediante la utilización de Juddi, los clientes ya no accederán directamente a los WS ofertados por el gestor. Ahora los clientes preguntarán a Juddi por un WS y será Juddi quien les diga la dirección.

Esto se hace porque los WS pueden cambiar de dirección, lo cual implicaría tener que modificar los clientes para reflejar este cambio de direcciones. Usando Juddi, da igual donde se encuentren nuestros WS, solo debemos cambiar la dirección en el registro de Juddi y los clientes ya sabrán a donde deben dirigirse para realizar las llamadas.

La configuración de Juddi en principio puede ser algo compleja y necesitaremos de la herramienta SoapUI. Básicamente deberemos crear un negocio que aglutine los WS que ofrecemos, crear un modelo y añadir los WS que ofrece el gestor. En este caso, Usuarios, Ventas e Histórico.

Necesitaremos además modificar los clientes para que pregunten a Juddi en sus llamadas a los WS en vez de realizar llamadas directas. Juddi les devolverá la dirección actual de los WS y los clientes podrán realizar las llamadas.

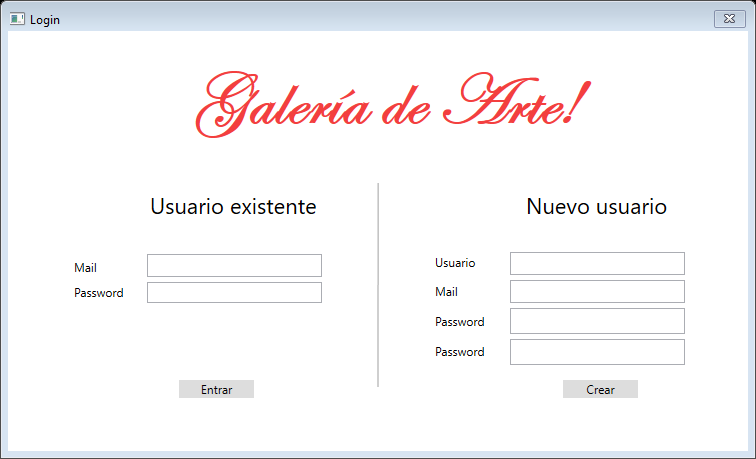
# Implementación del sistema

Como hemos comentado anteriormente, el proyecto consta de una serie de módulos, siendo cuatro de ellos, aplicaciones para el cliente. En los posteriores apartados se describirá el funcionamiento de cada una de estas aplicaciones.

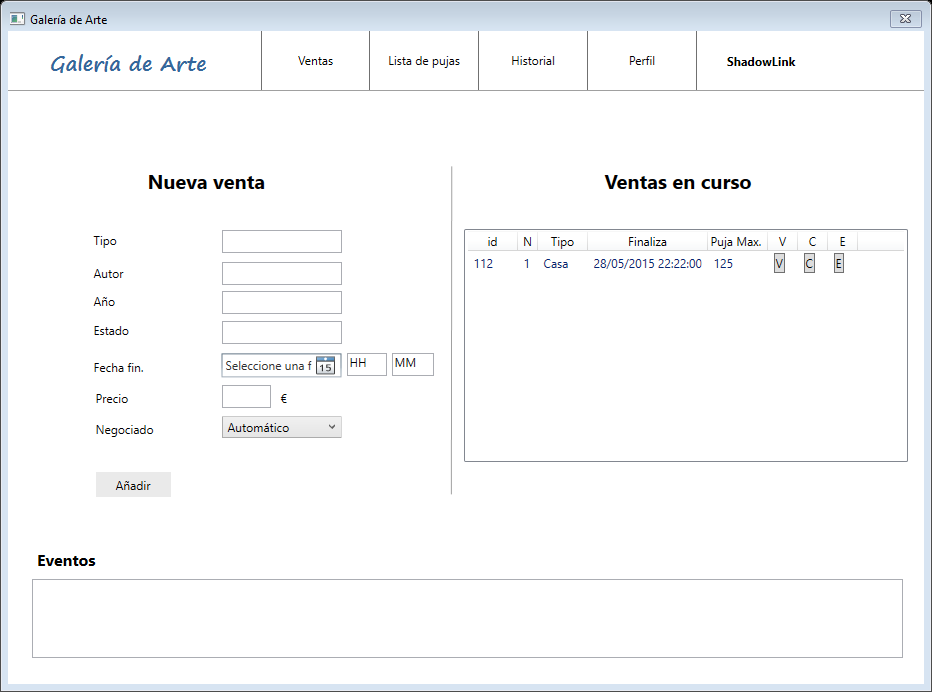
## Aplicación de escritorio Galería de Arte (.Net)

En este apartado hablaremos sobre la implementación en .Net de la aplicación para galerías de arte. Lo que se espera de la aplicación a grandes rasgos es lo siguiente:

* Registro de usuarios
* Función de autentificación
* Poder subastar obras de arte
* Poder pujar por obras de arte
* Historial de obras vendidas

A continuación hablaremos de forma pormenorizada de cada una de estas funcionalidades.

Arriba podemos ver la pantalla de inicio y registro de la aplicación. En ella podemos registrar un nuevo usuario o autentificarnos con uno ya existente.

Una vez autentificados entraremos en la aplicación principal. Como podemos observar, está compuesta por varias pestañas con los que podemos acceder a los diferentes apartados de la misma.

Como parte general podemos ver en la parte inferior el registro de eventos. Aquí nos irán apareciendo los avisos de subastas terminadas o canceladas.

Concretamente en la imagen podemos ver el apartado ventas. Podemos distinguir dos bloques. En el bloque de la izquierda encontramos el formulario donde poder registrar una nueva venta. En el podemos tenemos una serie d campos a rellenar como el tipo de obra, la fecha tope de subasta, el precio o si queremos negociado automático o manual.

Si seleccionamos negociado automático, el sistema escogerá la puja más elevada al finalizar el tiempo. En cambio, si seleccionamos negociado manual, será el vendedor quien deba escoger la puja ganadora de entre un listado de pujas.

En la parte de derecha de la pantalla encontramos un listado con las obras en venta en ese instante, estas obras se pueden encontrar en dos estados.

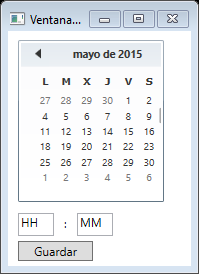
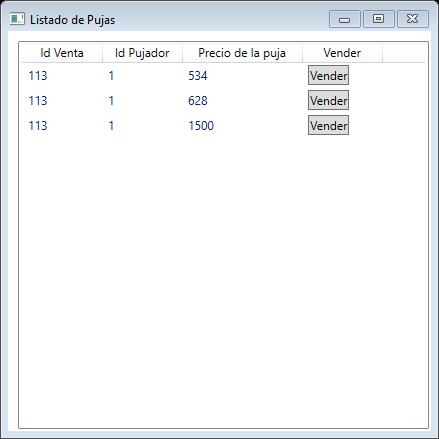
* En progreso
* Finalizadas

Cabe mencionar que una obra se encontrara en estado finalizada solo si se ha seleccionado el negociado manual. Esto quiere decir que la obra se encuentra a la espera de que el vendedor escoja una de las pujas. Una subasta automática que agota su tiempo desaparecerá automáticamente de la lista.

En el listado podemos encontrar entre otras cosas la fecha de finalización de cada subasta, la puja máxima en ese instante y tres controles:

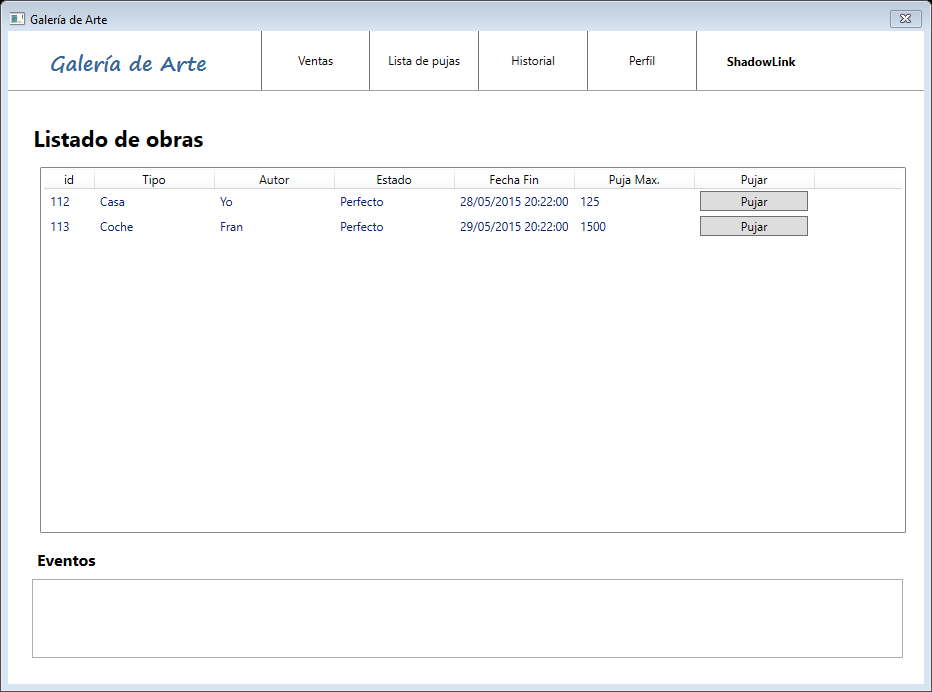
* V: Vender, se puede realizar en cualquier momento. Si el negociado es manual se desplegará la ventana de selección de pujas. Si el negociado es automático, se realizará la venta a la puja más alta en ese momento.
* C: Cancelar, la subasta se cancela y se elimina.
* E: Editar, abre una ventana que nos permite editar la fecha de finalización de la subasta.

A continuación veremos una muestra tanto de la ventana de edición como del listado de pujas.



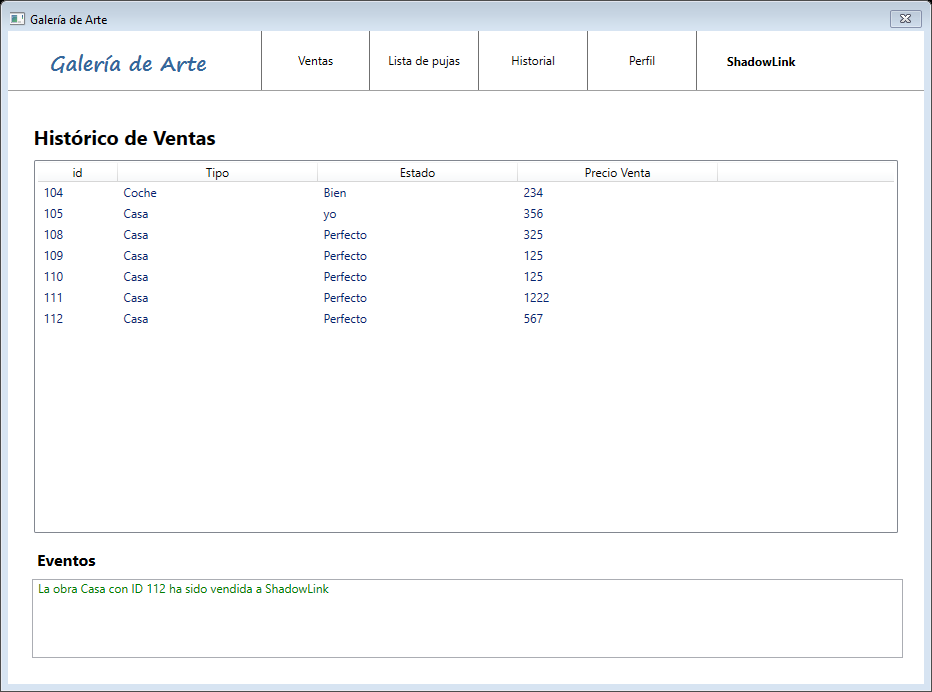
Como vemos, en el listado de pujas podemos ver las distintas pujas realizadas en ese momento sobre la obra en cuestión y seleccionar a que puja deseamos conceder la venta.

En cuanto a la ventana de edición, como ya hemos comentado, podemos cambiar la fecha de finalización de la subasta seleccionada.

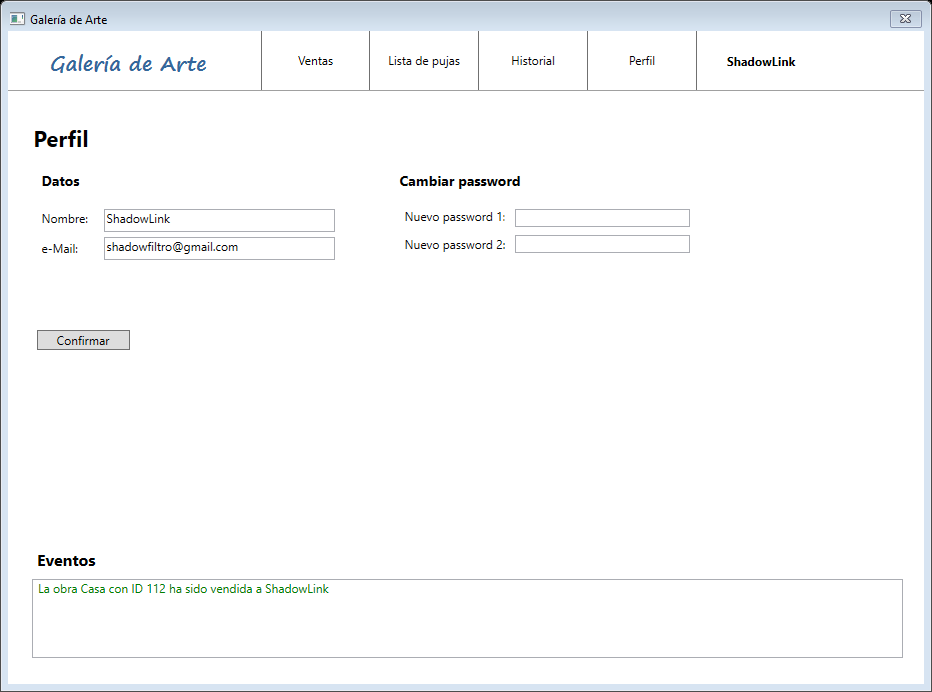


En la imagen superior encontramos la sección con el listado de obras en subasta en ese momento. Podemos ver en tiempo real cual es la puja más alta de cada subasta y pujar por ella. No podremos ofrecer una cantidad inferior a la puja más alta. Una vez la subasta haya terminado, desaparecerá automáticamente de este listado.

Cada vez que una venta finaliza, tanto el comprador como el vendedor son informados mediante e-Mail.



Lo que vemos arriba es el historial de ventas finalizadas. En el podemos ver cada una de las ventas que hemos finalizado y el precio por el cual han sido vendidas.

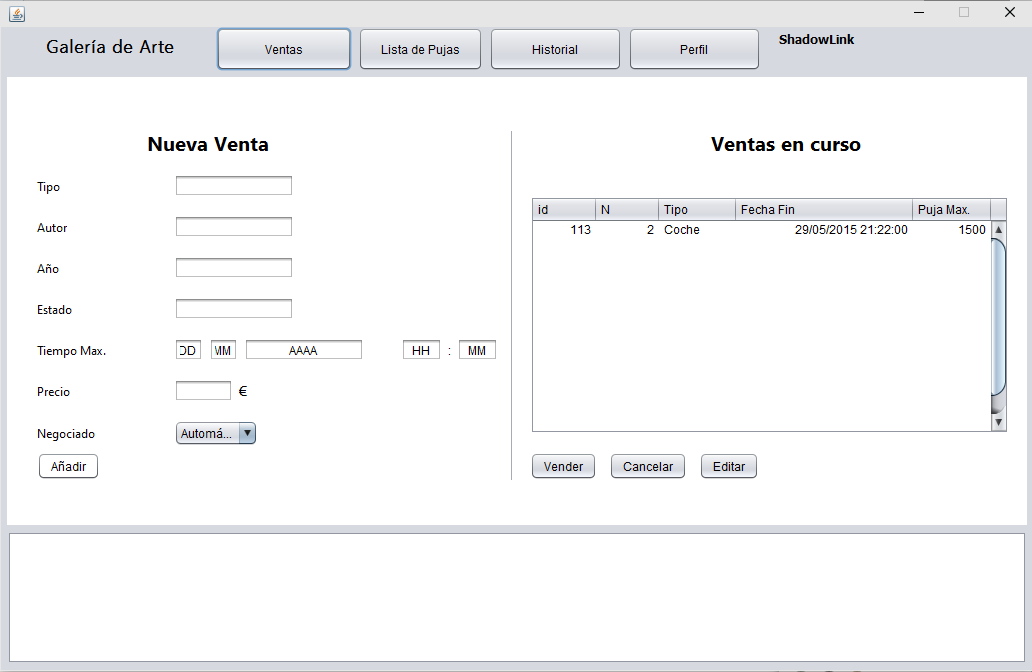
Como muestra, podemos ver que en el registro de eventos se ha generado una entrada por una obra que ha sido vendida en ese momento.

Por último nos encontramos con la página de perfil donde podemos editar los datos del usuario activo, nombre, e-Mail y contraseña.

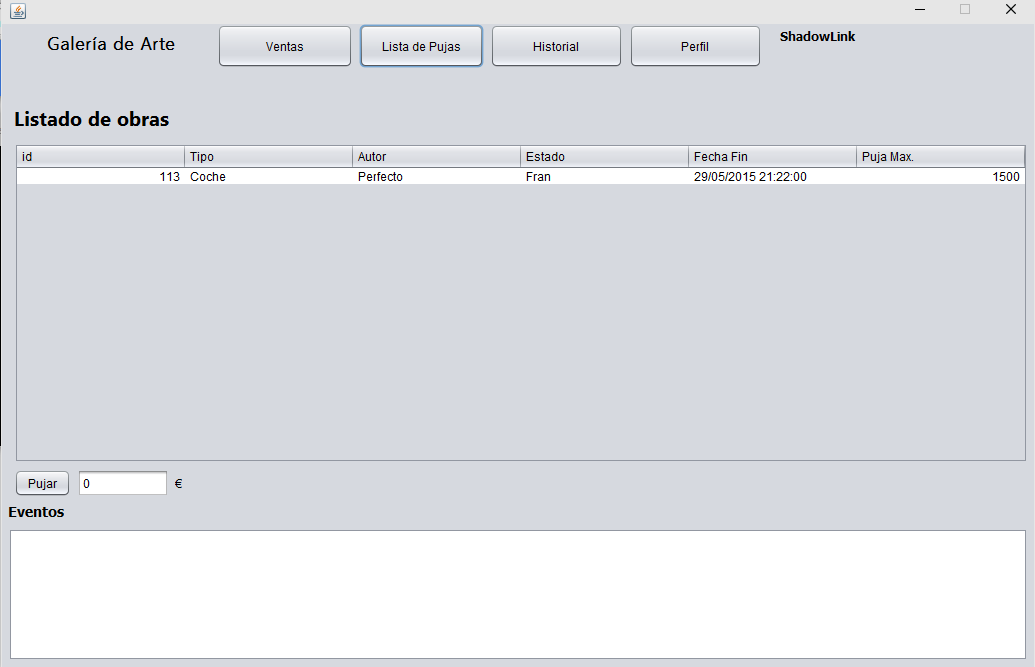
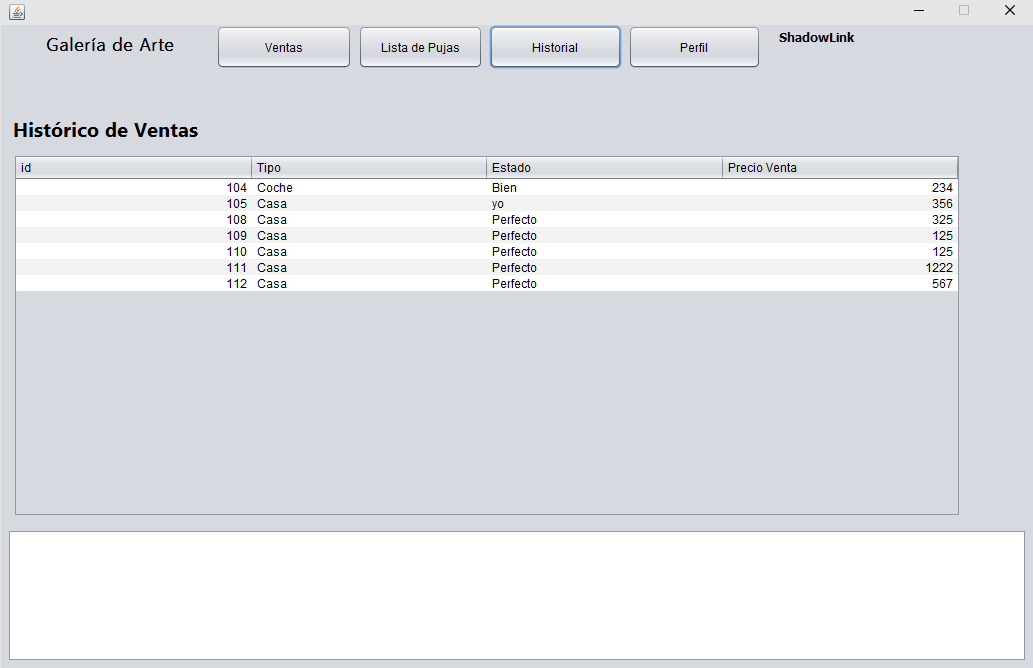
## Aplicación de escritorio Galería de Arte (Java)

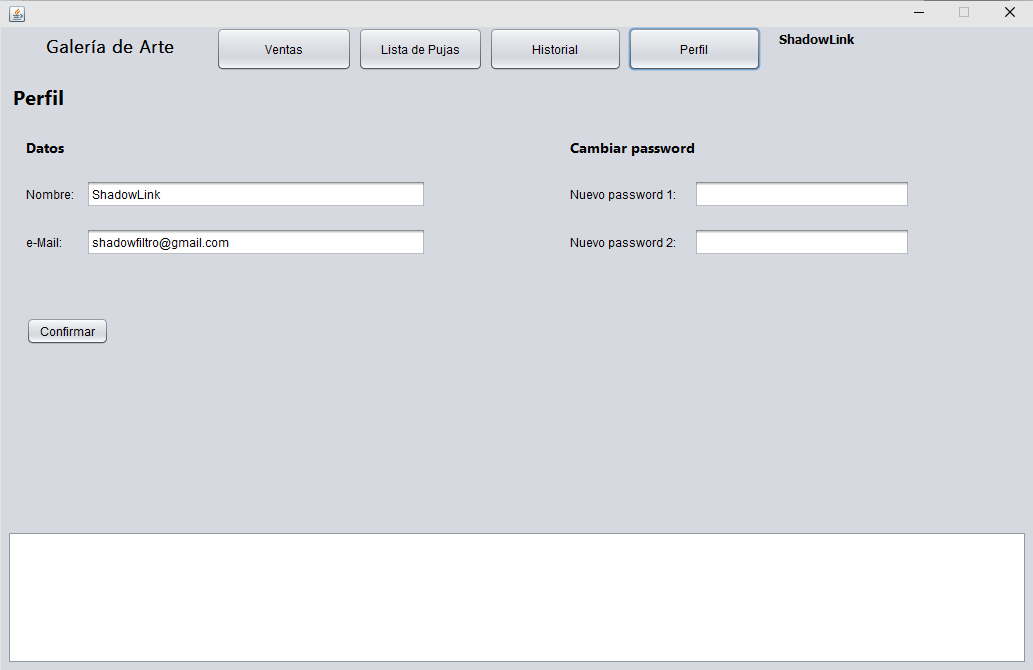
La aplicación para galerías de arte implementada en Java ese en esencia totalmente idéntica a la implementación en .Net salvo cambios menores. A continuación se ofrece una serie de imágenes que muestran su aspecto. El funcionamiento, como ya hemos dicho, es completamente idéntico a la implementación .Net salvo en el caso de algunos controles que han debido ser sustituidos.

**Pantalla de autentificación**

**Pantalla de ventas**

**Listado de obras en subasta**

**Histórico de ventas**

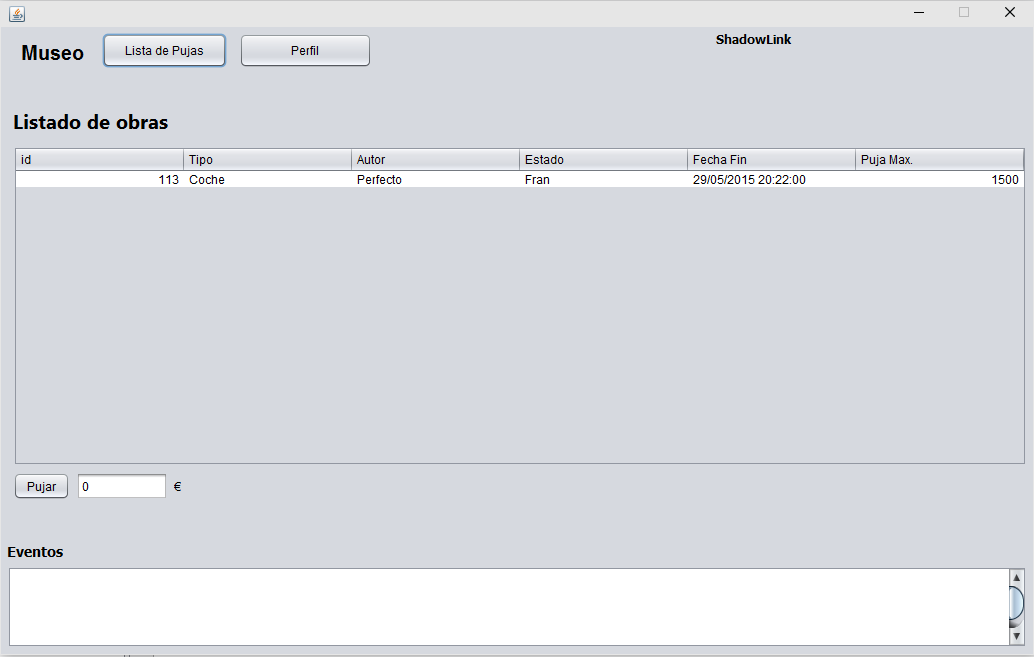
**Pantalla de perfil**

## Aplicación de escritorio para Museos (Java)

Los museos tienen la limitación de que solo pueden pujar por obras de arte y no vender. Por lo tanto la aplicación es una versión limitada en funciones de la aplicación para galerías de arte.

**Pantalla de autentificación**

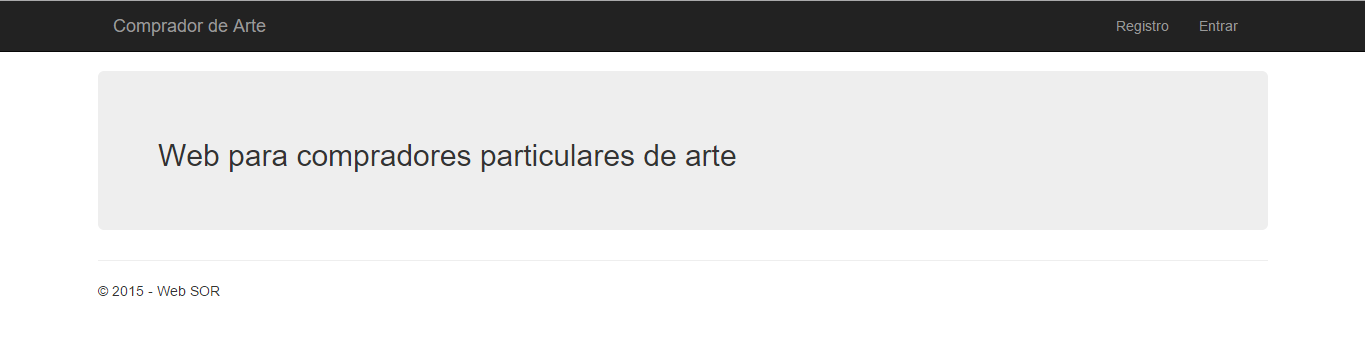
**Listado de obras en subasta**



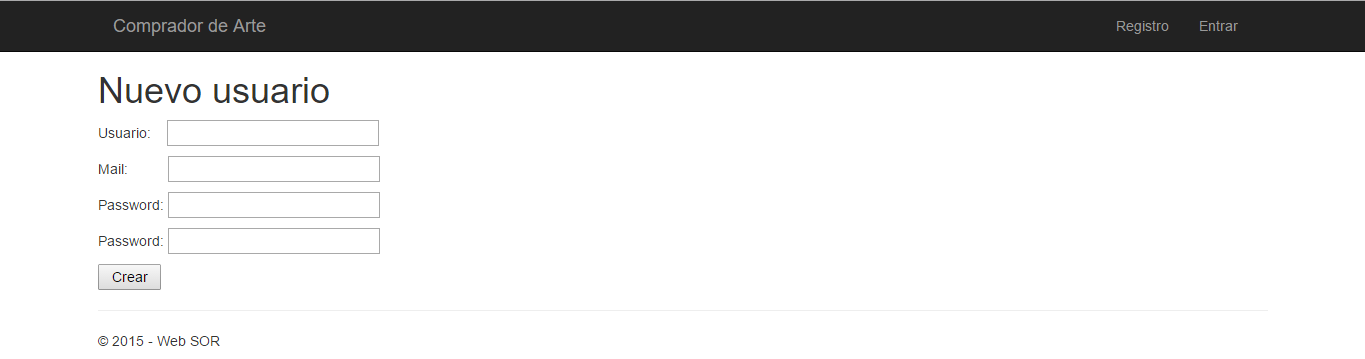
## Aplicación web (ASP)

La aplicación web esta enfocada a usuarios particulares que quieren vender sus obras de arte. Implementada en ASP, contiene las mismas funcionalidades que las aplicaciones de escritorio.

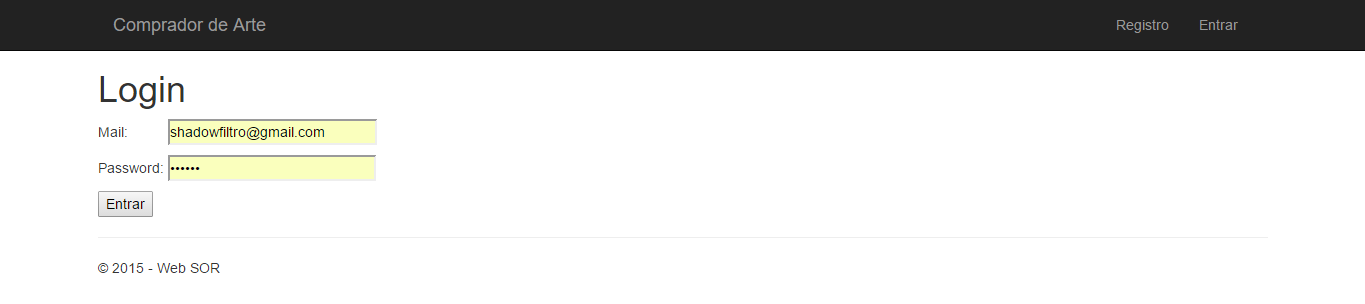
A continuación se ofrecen una serie de imágenes de esta aplicación que como ya hemos dicho provee las mismas funcionalidades.



**Registro de usuarios**



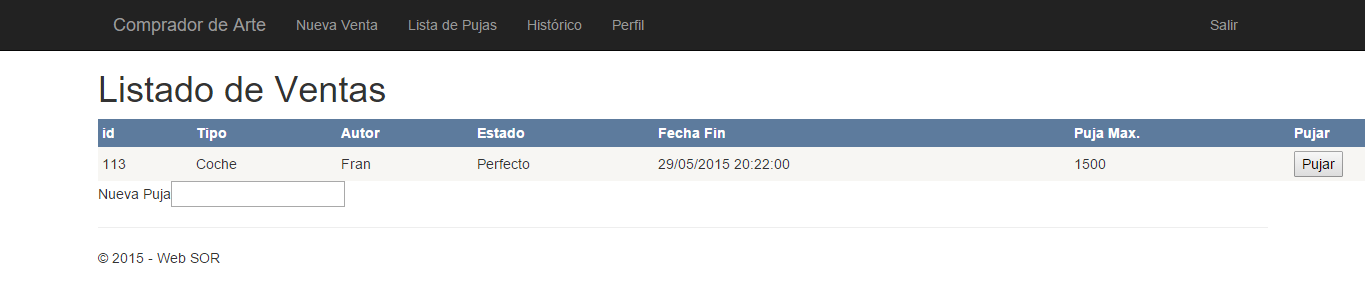
**Autentificación de usuarios**



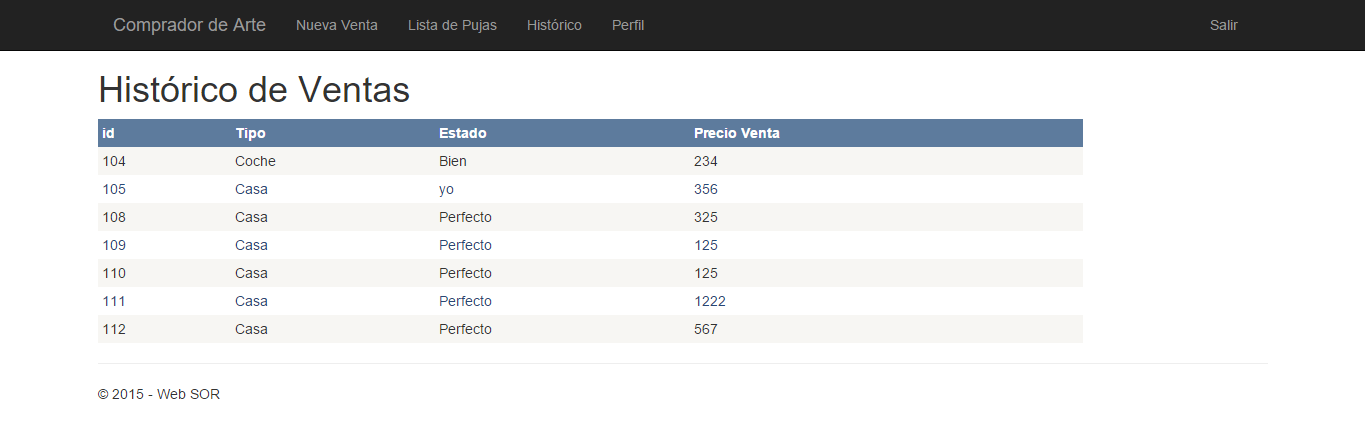
**Sección de ventas**



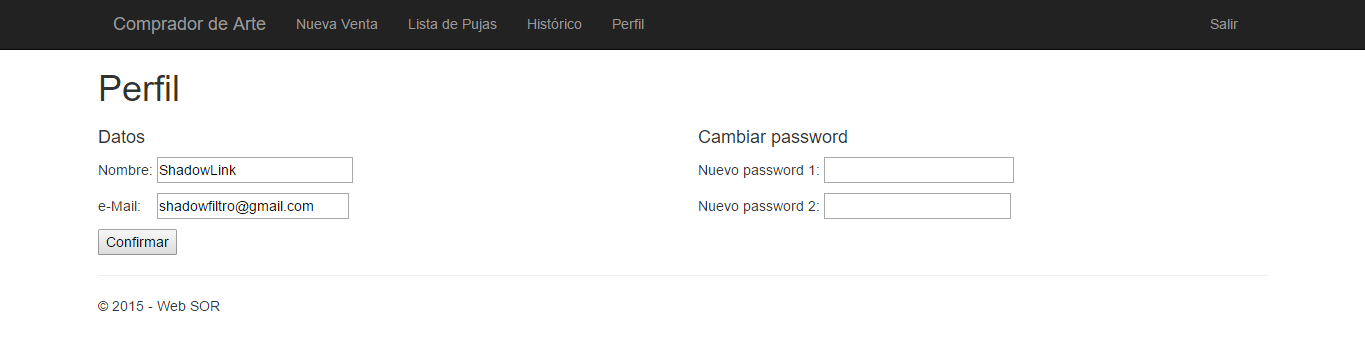
**Listado de obras en subasta**



**Histórico de ventas**



**Perfil**



## 

## Despliegue y configuración

Para poner en marcha el proyecto necesitas realizar una serie de pasos y configurar una serie de servicios. Vamos a tratar de explicar de forma esquematizada todo el proceso.

**Base de datos**

Debemos contar con una base de datos instalada en la máquina donde queramos poner en marcha el gestor. Para este proyecto se ha utilizado MariaDB, pero nos vale cualquier base de datos MySql. Una vez instalado el gestor de base de datos deberemos importar las tablas que se proporcionan en el fichero SOR.sql.

Con todo ello ya tendremos la base de datos lista.

**ActiveMQ**

Ahora deberemos poner en funcionamiento nuestro gestor de colas y topics. Si tenemos Java instalado solo debemos ejecutar el wrapper que encontraremos dentro del directorio de ActiveMQ. Con ello, ActiveMQ estará listo para aceptar conexiones, no debemos hacer ninguna configuración especial.

**Juddi**

El directorio de servicios Juddi es lo próximo que deberemos inicializar. Tan sencillo como ejecutar el script startup.bat. Puesto que los negocios, modelos y servicios ya están creados en el paquete que se proporciona, no debemos configurar nada más.

**Gestor Pujas**

Dentro del proyecto Gestor, encontraremos un proyecto llamado Gestor Pujas. Se trata de una aplicación de consola que gestiona las pujas entrantes entre otras cosas, por lo tanto será lo siguiente que debamos ejecutar.

**Internet Information Services (IIS)**

Será nuestro servidor web y lo necesitamos para servir tanto los webservices como la aplicación web para usuarios. Para instalarlo solo debemos agregarlo desde la opción “Activar o desactivar características de Windows” en “Agregar/quitar programas” del panel de control de Windows.

Una vez hecho esto, tendremos acceso al panel de administración de IIS. Debemos entonces desplegar las dos aplicaciones web mediante la opción “publish” de Visual Studio. Para ello, previamente debemos haber creado dos sitios nuevos desde el administrador de IIS donde poder desplegar estas aplicaciones.

Si todo ha funcionado conforme a lo esperado, tanto los webservices como la aplicación web de usuarios serán accesibles a través de la dirección y puerto que hayamos especificado.

**Clientes**

Ya solo nos queda ejecutar los clientes que necesitemos y acceder al sistema de pujas y todas las funciones que ofrece

**Esquema de red**

Mostramos a continuación un organigrama sencillo con una posible distribución de los diferentes módulos.



## Seguridad

Las opciones de seguridad del proyecto vienen dadas en varias vertientes, véase:

* **Contraseñas encriptadas:** Todas las contraseñas se almacena en la base de datos encriptadas mediante un algoritmo SHA-1. Cuando un usuario se autentifica en el servicio se obtiene la contraseña encriptada y se compara con la contraseña encriptada que el usuario ha proporcionado.
* **Log del sistema:** El gestor almacena en un log con fecha y hora todo lo que ocurre en el mismo. Altas, autentificaciones, pujas, ventas o cancelaciones entre otras cosas quedan registradas. Podemos encontrar este log en “C:\Users\usuario\AppData\Local\Temp/Sor.txt”.
* **Roles de usuario:** Los usuarios solo pueden autenticarse en las aplicaciones donde se hayan registrado. Así pues un usuario registrado a través de la web no podrá autentificarse en la aplicación para galerías de arte.

# Pruebas

**Formularios**

* Verificación de los datos introducidos en los formularios de cada una de las aplicaciones. Se muestran mensajes de error en el caso de que alguno de los campos no sea válido.

**Ventas**

* Puesta a la venta de artículos y propagación a todos los clientes.
* Las subastas acaban correctamente a la hora convenida. Si el negociado es automático, se ofrece un ganador de forma automática. Si el negociado es manual se espera a que el propietario del artículo valida una de las ofertas.
* La edición de fechas de artículos en subasta funciona correctamente, las subastas siguen acabando en la hora y fecha actualizada.
* La cancelación de subastas funciona correctamente en ambos negociados.
* La venta de artículos abre una ventana de pujas realizadas si el negociado es manual, o vende automáticamente si se trata de una subasta con negociado automático.
* Si una subasta acaba sin pujas se declara desierta y se comunica a todos los clientes conectados.

**Pujas**

* Solo las pujas superiores al precio actual se toman en cuenta.
* Si un usuario puja dos veces, solo se mantiene la puja más alta.
* Una vez agotado el tiempo de subasta, el artículo desparece del listado de ventas para evitar pujas fuera de plazo.
* Una vez una subasta ha finalizado, se comunica a todos los clientes conectados el ganador y el artículo.

**Login**

* No se permite la autenticación de usuarios registrados con otro nivel de seguridad.
* El registro de usuario funciona perfectamente.

# 

# Valoración del proyecto

En conjunto ha sido un proyecto interesante para conocer un conjunto de tecnologías menos conocidas, al menos para mí, como son Juddi o ActiveMQ pese a lo complicado de encontrar documentación de las mismas.

Poder interconectar sistemas desarrollados en lenguajes distintos, lidiando con posibles incompatibilidades ha resultado ser una tarea reconfortante, sobre todo cuando todo empieza a encajar y funcionar.

Por contra pienso que es demasiado extenso para una sola persona al estar pensado inicialmente para grupos. Ciertas tareas como la doble implementación en Java y .Net del mismo cliente se hace bastante repetitiva una vez se han resuelto los pequeños aspecto diferenciadores, por ello pienso que la aplicación .Net y Java debería ser distintas de algún modo o aportar funcionalidades diferentes, o directamente eliminar una de ellas si el proyecto lo está desarrollando una sola persona como ha sido mi caso. Es interesante mantener dos aplicaciones que utilicen dos lenguajes distintos por ello pienso que debería eliminarse la aplicación Java de galería de arte y ampliar las funcionalidades de la aplicación de museos.