Trabajo Colaborativo: Unidad Didáctica 6

Gestión de una tienda virtual

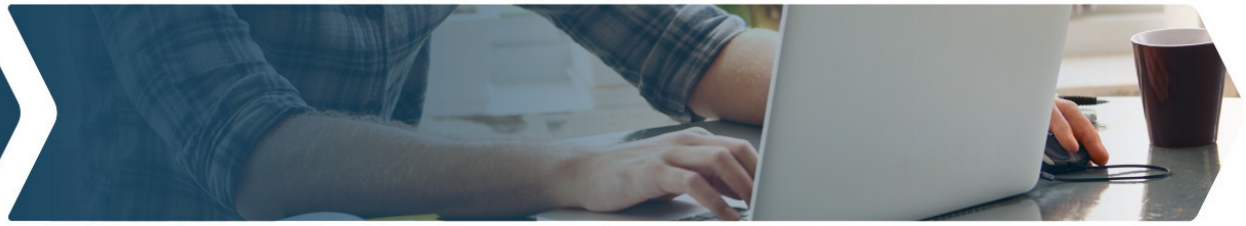
Alumnos: María A. , Pablo Enrique Guntín Garrido, Pablo M. , Pilar P.

Asignatura Programación Concurrente y Distribuida

Grado en Ingeniería Informática

curso 2024-2025

Índice



1. Enunciado

El dueño de un comercio dedicado a la venta de artículos diversos, ha observado que el número de clientes tradicionales ha disminuido, sin embargo, las operaciones a través de la red han ido creciendo significativamente en los últimos años. En este escenario, y para poder recuperar la salud del negocio, decide crear una réplica virtual de su tienda de manera online. Para poder facilitar la tarea del servicio web de gestión, nos pide que creemos tantos objetos remotos como sean necesarios para implementar la lógica básica de la tienda.

A continuación, se detallan los requisitos de las funcionales de nuestro software remoto:

Interfaz del objeto remoto:

1. doUserLogin:

a. Los usuarios que tienen permiso para acceder al sistema estarán almacenamos en un archivo JSON (formato clave-valor) al que tendrá que conectarse el objeto remoto.

2. registerNewUser:

a. Si un usuario no está registrado podrá registrarse. Para poder realizar el alta de un nuevo usuario se solicitará:

b. Nombre de usuario. Debe ser alfanumérico (solo letras y dígitos)

c. Password. Mínimo 8 caracteres, al menos una mayúscula, mínimo un dígito y algún carácter especial: *, & o %.

d. Email.

e. Confirmación de email. Debe ser igual que email. En caso de ser diferentes se informará con un mensaje.

f. Teléfono. Al menos con 9 dígitos.

3. insertProductInShop:

a. Se pueden crear nuevos productos en la tienda. Cada producto dispondrá de un nombre y un precio que habrá que almacenar en el archivo JSON.

4. showProductsInShop:

a. Se pueden mostrar todos los productos de la tienda, contenidos en el archivo JSON.

5. deleteProductoInShop:

a. Se puede borrar un determinado producto de la tienda, indicando su identificador.

Esta práctica se realizará en dos partes:

1. **Proyecto de objetos remotos** que implementan toda la lógica especificada anteriormente, incluyendo la conexión con el archivo JSON.

2. **Proyecto de pruebas**:

    a. Tendrá un menú por línea de comandos que permitirá invocar a cada una de las tareas que permite el objeto remoto.

b. Los resultados se mostrarán también por línea de comandos.

1.1 Introducción

Este trabajo colaborativo tiene como objetivo el desarrollo de una aplicación de tienda remota utilizando la tecnología RMI (Remote Method Invocation) en Java.

El desarrollo de este trabajo es la creación de un sistema de tienda virtual que pueda ser accedido de manera remota. La aplicación consta de dos componentes principales: el **servidor** y el **cliente**.

El servidor gestiona el proceso de login de usuarios, registro de los nuevos y la gestión de productos (inserción y eliminación) en una tienda virtual.

El cliente se conecta al servidor de manera remota, invocando métodos a través de RMI para acceder a los productos, registrarse como nuevo usuario o iniciar sesión en el sistema.

En una tienda virtual, la interacción con los productos se realiza de manera digital, y los clientes deben acceder a un sistema para gestionar productos, realizar compras y gestionar sus cuentas de usuario. RMI facilita que el cliente pueda invocar de forma remota las operaciones del servidor, como puede ser el registro de un nuevo usuario o la consulta de productos disponibles. Esto ofrece ventajas de escalabilidad y de distribución de tareas.

Una de las principales ventajas es la capacidad de desacoplar el servidor y el cliente. El servidor maneja toda la lógica de negocio y almacenamiento de datos, mientras que el cliente solo se encarga de interactuar con el usuario y enviar solicitudes al servidor. Este modelo de comunicación remota basado en RMI permite una interacción eficiente y flexible.

El trabajo se desarrolla bajo la premisa de un entorno de pequeña escala, donde la persistencia de datos se maneja a través de un archivo JSON. Este archivo almacena los datos de los usuarios y los productos de la tienda, lo que permite simular una base de datos ligera para pruebas. El servidor es responsable de gestionar la autenticación de usuarios, la creación de nuevas cuentas y la manipulación de los productos disponibles en el sistema. La parte del cliente no se implementa de forma completa, tan solo se crea una interfaz sencilla que permita probar las funciones del servidor.

Se implementan varias funcionalidades clave:

Login de usuarios: Permite a los usuarios registrados acceder al sistema de acuerdo con sus credenciales almacenadas en el servidor.

Registro de nuevos usuarios: Los usuarios no registrados pueden crear nuevas cuentas, proporcionando su nombre de usuario, correo electrónico y número de teléfono.

Gestión de productos: Los productos pueden ser añadidos, eliminados o visualizados en la tienda virtual.

2. Descripción del problema

El dueño de una tienda física quiere expandir su negocio al ámbito digital, implementando una plataforma de ventas en línea. Para ello, se requiere un sistema que permita gestionar usuarios, productos, y realizar compras de manera remota a través de un cliente que se conecte con un servidor. El desafío principal es la comunicación remota entre el cliente y el servidor usando la

tecnología RMI en Java, asegurando que las acciones realizadas en el cliente se reflejen correctamente en el servidor y en la base de datos.

3. Solución implementada

El trabajo colaborativo consiste en implementar una tienda remota usando la tecnología Remote Method Invocation (RMI) en Java.

RMI permite comunicarse remotamente entre un cliente y un servidor. El cliente invoca métodos en el server como si estuviera en la misma máquina.

Para ello, hemos distribuido el trabajo en dos componentes principales, el servidor y el cliente. Tal y como se solicita en el enunciado, cada uno de estos componentes es un proyecto separado.

Puesto que parte del enunciado exige almacenar los datos de usuarios y productos en un archivo JSON y Java no incluye funciones para trabajar con archivos de este tipo de forma nativa, el proyecto servidor debe incluir una librería externa. En nuestro caso, optamos por la librería Jackson Project por ser una de las más conocidas y que más tiempo llevan en circulación.

3.1 Proyecto servidor

La funcionalidad del servidor tiene una clara separación lógica entre las funciones asociadas a los usuarios y aquellas relativas a los productos. Por tanto decidimos separar dichas funcionalidades también a nivel de código

3.1.1 Gestión de usuarios

La parte de gestión de usuarios tiene tres partes a nivel de código. Una interfaz para definir las funciones, una clase para implementar las funciones de la interfaz y una clase para definir el objeto usuario como tal.

3.1.1.1 Interfaz *InterfaceUsuarios*

La funcionalidad de gestión de usuarios se define mediante la interfaz *InterfaceUsuarios*, que define las siguientes funciones

doUserLogin

Esta función toma como parámetros el usuario y contraseña, y tras comprobarlos con los datos almacenados, devuelve “True” o “False” dependiendo de si la autenticación ha tenido éxito o no.

registerNewUser

A esta función se le pasan como parámetros todos los datos del formulario de registro de usuarios (usuario, contraseña, email, email repetido y teléfono) y la función realiza una validación del formato de los mismos antes de crear el usuario.

En caso de que se produzca un error de validación, devuelve un string con los errores detectados. Si todo va bien, el string está vacío.

Desde un punto de diseño, en lugar de devolver texto con los errores de validación sería más correcto definir un sistema de códigos de error, pero consideramos que era excesivo para un proyecto de esta escala.

3.1.1.2 Clase Usuarios

La clase Usuarios extiende la clase UnicastRemoteObject para poder utilizarse en una arquitectura RMI e implementa la clase *InterfaceUsuarios.*

Además de las funciones de la interfaz, esta clase incluye las siguientes partes:

- Una lista de objetos *Usuario* para almacenar la lista de usuarios en memoria durante la ejecución y poder trabajar con ella.
- Un string donde almacenar la ruta del archivo JSON en donde se guardan los usuarios.
- Un método creador que toma como parámetro la ruta al archivo JSON. Este método lee el archivo JSON e inserta los usuarios en la lista. En caso de error al leerlo, se comienza con un listado vacío.

3.1.1.3 Clase Usuario

Esta clase define las propiedades de un Usuario, con sus funciones Get asociadas. La excepción es la contraseña, que no se puede obtener de forma directa por seguridad. En su lugar, se define el método *checkPassword*, que comprueba si la contraseña que se le pasa por parámetro es correcta o no.

3.1.2 Gestión de productos

Al igual que la parte de los usuarios, la gestión de los productos también incluye una interfaz, una clase para implementarla y una clase de tipo objeto.

3.1.2.1 Interfaz *InterfaceUsuarios*

En este caso, se definen las siguientes funciones:

insertProductstInShop

Permite insertar un nuevo producto a la tienda pasando como parámetro el nombre y el precio. Devuelve “True” en caso de éxito o “False” si se produce un error.

deleteProductsInShop

Permite eliminar un producto de la tienda en base a su ID. Dicho ID se toma como la posición en la lista de productos.

showProductsInShop

Esta función devuelve una lista de objetos de tipo Producto. Es el propio cliente el que luego tiene que procesar dicha lista para mostrarla adecuadamente.

3.1.2.2 Clase Productos

La clase Productos extiende la clase UnicastRemoteObject para poder utilizarse en una arquitectura RMI e implementa la clase *InterfaceProductos.*

Además de las funciones de la interfaz, esta clase incluye las siguientes partes:

- Una lista de objetos *Producto* para almacenar la lista de usuarios en memoria durante la ejecución y poder trabajar con ella.
- Un string donde almacenar la ruta del archivo JSON en donde se guardan los productos.
- Un método creador que toma como parámetro la ruta al archivo JSON. Este método lee el archivo JSON e inserta los productos en la lista. En caso de error al leerlo, se comienza con un listado vacío.

3.1.2.3 Clase Producto

Esta clase define las propiedades de un producto. Puesto que se utiliza como salida de la función *showProductsInShop,* es necesario que implemente la interfaz *Serializable*

3.1.3 Clase principal del servidor

La clase principal del servidor, llamada ServidorTienda requiere los siguientes parámetros al ejecutarla por la línea de comandos:

- IP a la que asociar la interfaz RMI
- Puerto TCP en el que escuchar
- Ruta del archivo JSON para almacenar usuarios
- Ruta del archivo JSON para almacenar productos

Con todos los parámetros, el servidor instancia una clase Usuarios, una Productos y las asocia a rmi://<ip>:<puerto>/usuarios y rmi://<ip>:<puerto>/productos respectivamente

4. Pasos a seguir

4.1 Configuración inicial

A parte de tener instalado un entorno de desarrollo, en nuestro caso Eclipse, e instalado JDK adecuado (Java 8 o superior, depende de cada miembro del equipo de trabajo).

Es necesario tener bien estructurado el proyecto. En nuestro caso, tenemos dividido el proyecto en dos carpetas: Pruebas y Tienda.

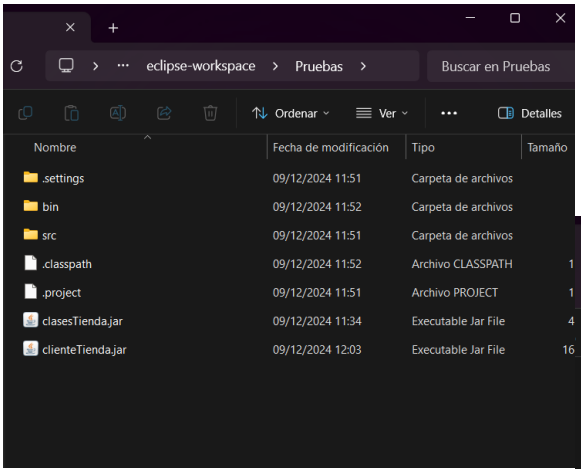
Dentro de pruebas tenemos ClienteTienda.jar y ClasesTienda.jar.

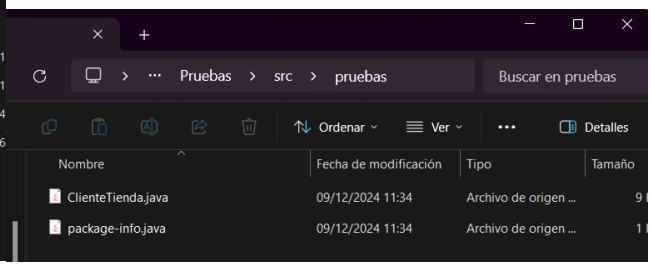
Dentro de Tienda, tenemos los archivos .jar correspondientes y los archivos JSON necesarios para realizar el trabajo, que se encuentran en la raíz:

productos.json

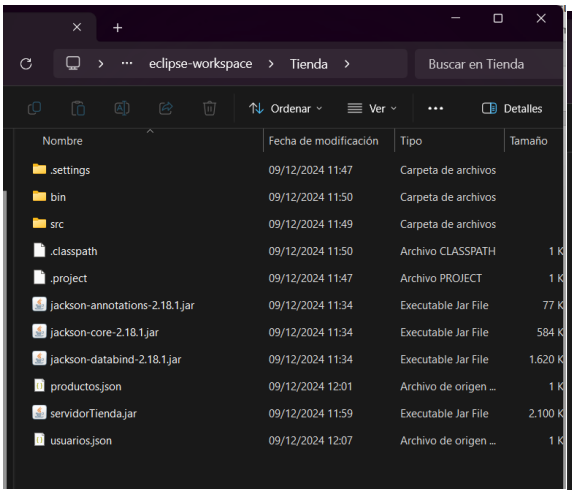
usuarios.json

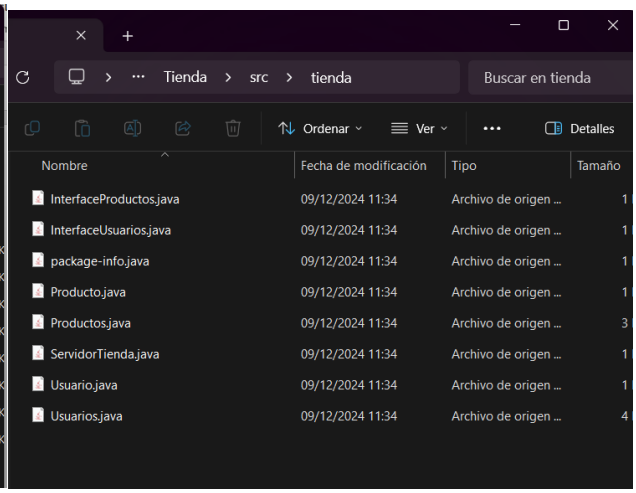
Pruebas:

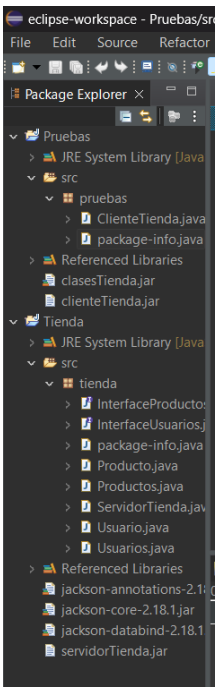




Tienda:







Para lo puertos, como se explica el punto 3.1.3, se especifica en la línea de comandos del terminal a la hora de ejecutar el proyecto. Se puede usar todo puerto superior al 1024.

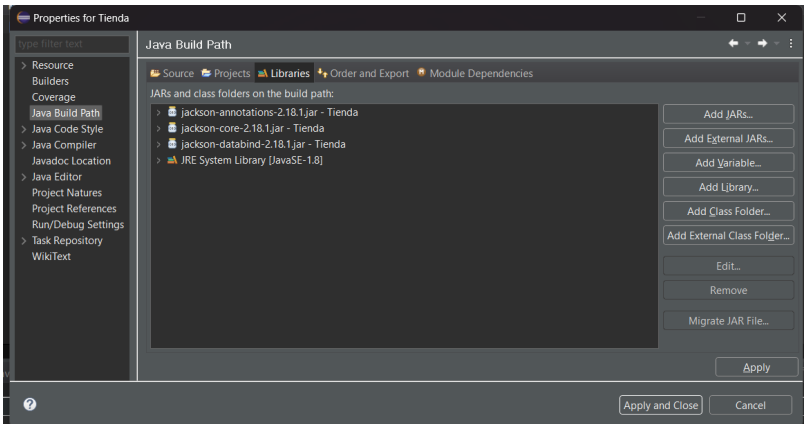
4.2 Implementación

Antes de todo se debe compilar cada proyecto, tanto el de Pruebas como el de Cliente.

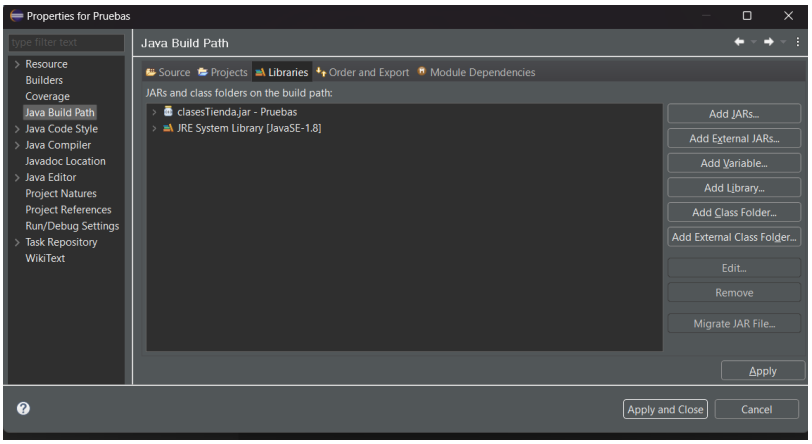
Primero, se adjuntas los archivos jar en Tienda y Pruebas:

Clic derecho sobre cada proyecto → Propierties → Java Build Path → Libraries → Add JARs…

Tienda:



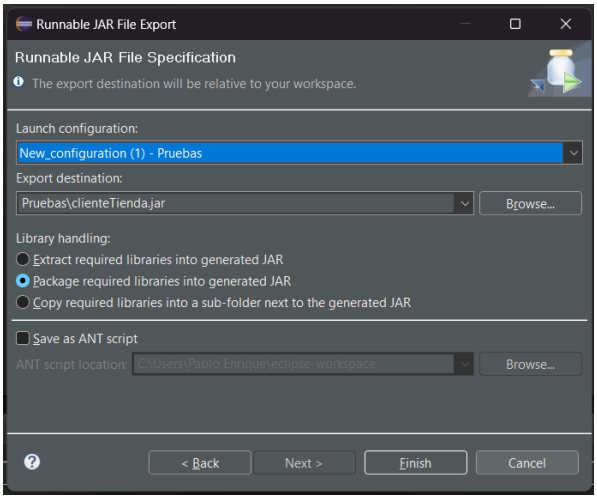
Pruebas:



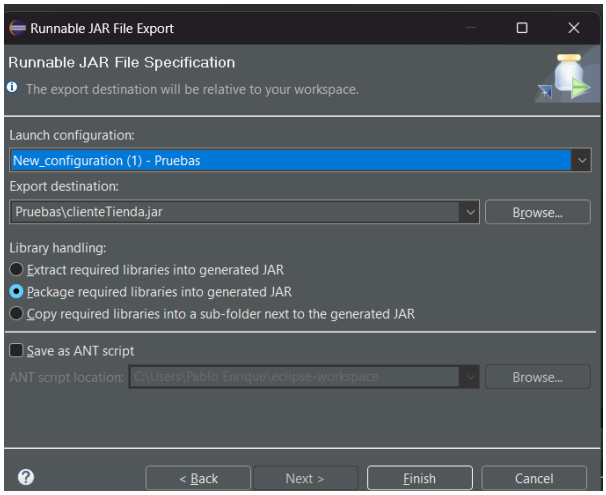
Lo siguiente es compilar ambos proyectos, para ello:

Clic derecho en cada uno → Export → Runnable JAR file → Configurar cada uno con sus parámetros:

Pruebas:



Tienda:



4.2 Ejecución en el sistema

Para cada proyecto, abrimos dos terminales para ejecutar en cada uno cada proyecto.

Primero, ejecutamos Tienda desde el directorio raíz del mismo proyecto:

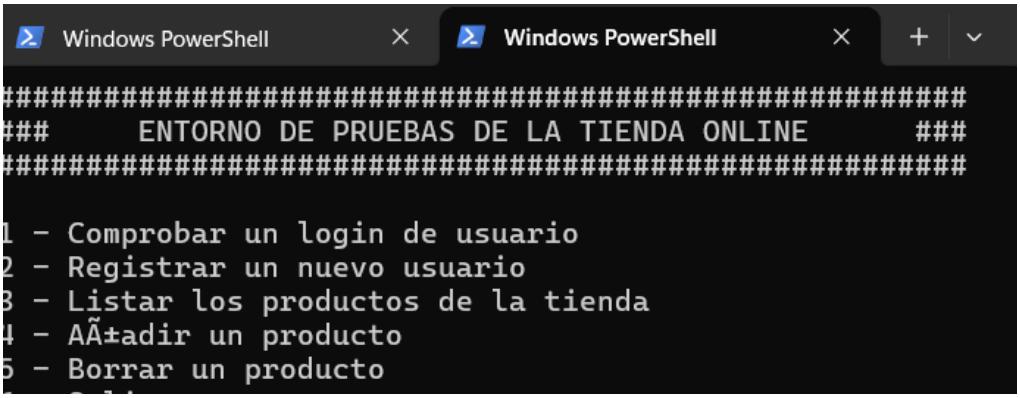
```
... 2 more
PS C:\Users\Pablo Enrique\eclipse-workspace\Tienda> java -jar servidorTienda.jar localhost 1025 usuarios.json productos.
json
No se han podido cargar usuarios. Se comienza con un listado vacio
No se han podido cargar productos. Se comienza con un listado vacio
Servidor listo
```

Se puede ver como el servidor está listo.

Ahora, se debe ejecutar Pruebas desde el directorio raíz del mismo proyecto:

```
PS C:\Users\Pablo Enrique\eclipse-workspace\Pruebas> java -jar clienteTienda.jar localhost 1025
```

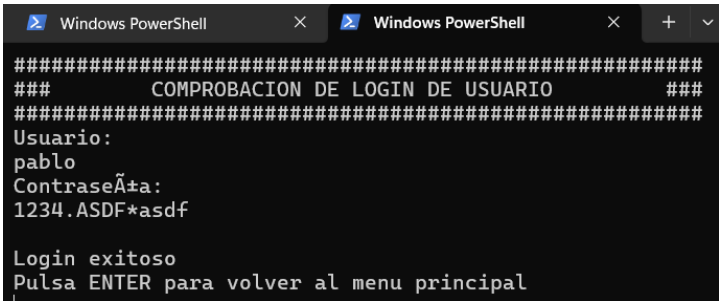
Una vez al ejecutar este comando ya podemos operar con la aplicación:



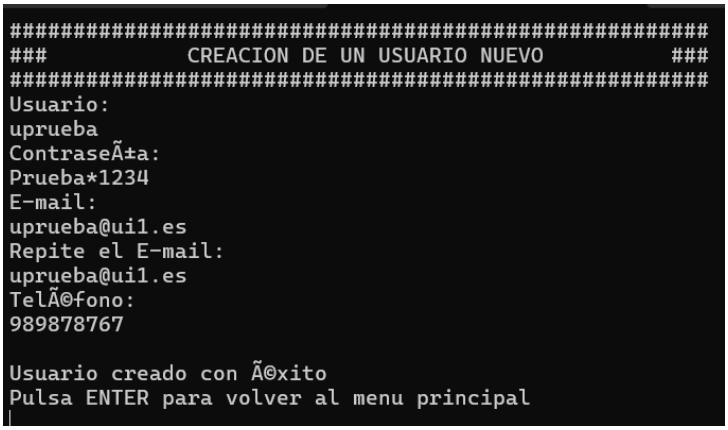
5. Resultados

Después de haber ejecutado el proyecto en el apartado anterior, ahora se muestran los resultados:

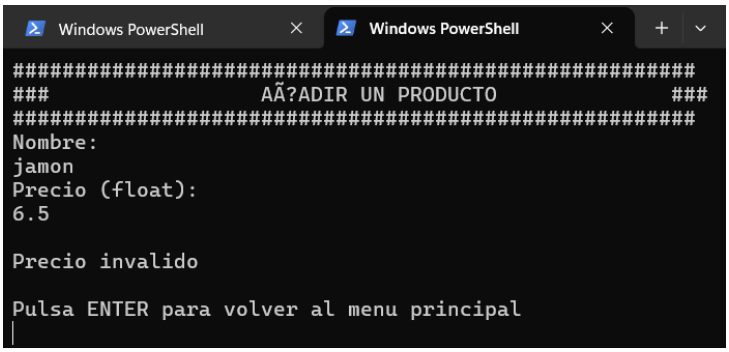
Comprobar usuarios:



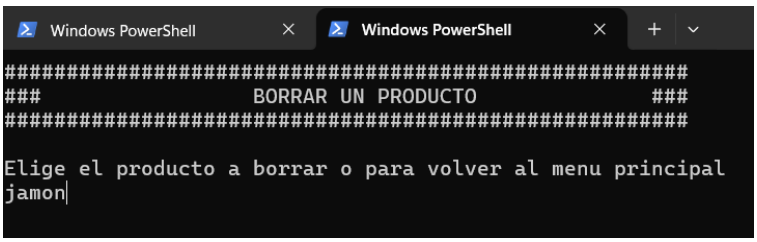
Registrar usuarios:



Añadir producto:



Eliminar producto:



Todo ello se actualiza en los archivos JSON creados para ello, para usuarios y para productos.

6. Conclusión

Mediante la realización de esta actividad "Gestión de una tienda virtual" usando para ello, una implementación basada en RMI, hemos logrado comprender el mecanismo de las comunicaciones remotas en Java.

En estas comunicaciones resulta esencial que exista una interacción cliente-servidor en la que toman protagonismo por un lado el servidor, quién proporciona los servicios, y por otro, el cliente que es quién realiza las peticiones al servidor.

Debemos tener en cuenta, 3 principios fundamentales en la interacción cliente-servidor: el encapsulado de los servicios, el acceso transparente y la persistencia de datos.

En el caso del encapsulado de los servicios, estos quedan integrados en objetos que implementan una interfaz remota común. En el caso de estudio, la interfaz ShopInterface, define los métodos con los que el cliente puede interactuar con el servidor, establece el contrato común (asegurando que el servidor y el cliente "hablen el mismo idioma") y proporciona una manera estandarizada de invocar los métodos remotos.

Además, el cliente no necesita conocer detalles como la ubicación del servidor ni cómo se serializan los datos (acceso transparente), ya que la gestión de estos aspectos recae en el sistema RMI, permitiendo al cliente interactuar con el servidor de una manera sencilla.

Tras las diferentes pruebas realizadas en la elaboración del proyecto, hemos podido comprobar que otro pilar importante son los datos, tanto de usuarios como productos (Persistencia de los datos). Esta persistencia, mediante el almacenamiento en un archivo JSON, permite que los cambios realizados se conserven de manera permanente, evitando de esta forma comenzar de nuevo (desde 0) o introducir nuevos datos en las nuevas interacciones.

7. Bibliografía

[1] «Remote Method Invocation in Java» *Geeksforgeeks* [En línea]. Disponible en: https://www.geeksforgeeks.org/remote-method-invocation-in-java/ [Accedido: 30-nov-2024].

[2] «Invocación Remota de Métodos (RMI)» *Programacion.net* [En línea]. Disponible en: https://programacion.net/articulo/invocacion-remota-de-metodos-rmi_107/2 [Accedido: 30-nov-2024].

[3] C. Crescas «Multithreaded Server with Java RMI» *Medium, 2022* [En línea]. Disponible en: https://colecrescas.medium.com/multithreaded-server-with-java-rmi-ced7f8eedd94 [Accedido: 30-nov-2024].

[4] I. Claver «Procesos y servicios. RMI (I)» *Programandoapasitos* [En línea]. Disponible en: https://www.programandoapasitos.com/2016/01/procesos-y-servicios-rmi-i.html?m=1 [Accedido: 4-dic-2024].

[5] «Definición de RMI (Java RMI)» *Alegsa* [En línea]. Disponible en: https://www.alegsa.com.ar/Dic/rmi.php#gsc.tab=0 [Accedido: 4-dic-2024].

[6] «Java RMI» *Departamento de Informática UEDPG* [En línea]. Disponible en: https://deinfo.uepg.br/~alunoso/2017/RMI/ [Accedido: 4-dic-2024].

[7] «Java RMI» *Runnablepatterns, 2019* [En línea]. Disponible en: https://runnablepatterns.com/languages/java/java-rmi/ [Accedido: 4-dic-2024].

[8] «Ejemplo de Java Remote Method Invocation | RMI» *Parzibyte* [En línea]. Disponible en: https://parzibyte.me/blog/2018/02/12/ejemplo-java-rmi/ [Accedido: 4-dic-2024].

[9] «RMI y CORBA: Una base para aplicaciones Distribuidas» *Osmosislatina* [En línea]. Disponible en: https://www.osmosislatina.com/java/rmi.htm#marsh [Accedido: 5-dic-2024].

[10] «Ejemplo de RMI en Java Tienda Online con persistencia JSON y Programación distribuida cliente servidor» *AJPDSoft* [En línea]. Disponible en: https://proyectoa.com/ejemplo-de-rmi-en-java-tienda-online-con-persistencia-json-y-programacion-distribuida-cliente-servidor/ [Accedido: 5-dic-2024].