|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | Escuela Politécnica Superior  Ingeniería Informática  Prácticas de Sistemas Informáticos 2 | | | |
| **Grupo** | **2402** | **Práctica** | 1B | **Fecha** | 08/03/2021 |
| **Alumno/a** | | Rivas Molina, Lucía | | | |
| **Alumno/a** | | Santo-Tomás López, Daniel | | | |

# Práctica 1B: Arquitectura de JAVA EE

## Cuestión número 1:

**Abrir el archivo VisaDAOLocal.java y comprobar la definición de dicha interfaz. Anote en la memoria comentarios sobre las librerías Java EE importadas y las anotaciones utilizadas. ¿Para qué se utilizan? Comparar esta interfaz con el fichero de configuración del web service implementado en la práctica P1A.**

Están importadas las librerías java.sql para el acceso a la base de datos:

* java.sql.connection para establecer la conexión con la base de datos.
* Java.sql.resultset para almacenar los resultados de las consultas.
* Java.sql.sqlexception con las excepciones.
* Java.sql.statement para ejecutar sentencias.
* Javax.ejb.local para indicar que es local.

## Ejercicio 1:

**Introduzca las siguientes modificaciones en el bean VisaDAOBean para convertirlo en un EJB de sesión stateless con interfaz local:**

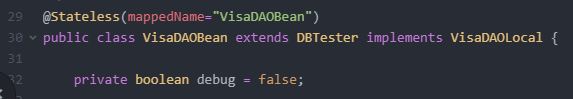
** Hacer que la clase implemente la interfaz local y convertirla en un EJB stateless mediante la anotación Stateless**

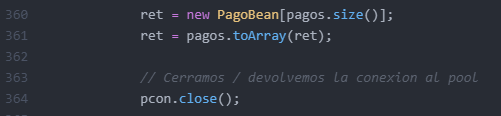
** Eliminar el constructor por defecto de la clase.**

** Ajustar el método getPagos() a la interfaz definida en VisaDAOLocal**

** Incluye en la memoria cada fragmento de código donde se han ido añadiendo las modificaciones solicitadas.**

Hemos añadido @stateless en vez del webservice en la clase VisaDaoBean.java y en el método getPagos() hemos vuelto a devolver arrays en vez de arraylists.





## Ejercicio 2:

**Modificar el servlet ProcesaPago para que acceda al EJB local. Para ello, modificar el archivo ProcesaPago.java de la siguiente manera: (…).**

**Importante: Esta operación deberá ser realizada para todos los servlets del proyecto que hagan uso del antiguo VisaDAOWS. Verifique también posibles errores de compilación y ajustes necesarios en el código al cambiar la interfaz del antiguo VisaDAOWS (en particular, el método getPagos()).**

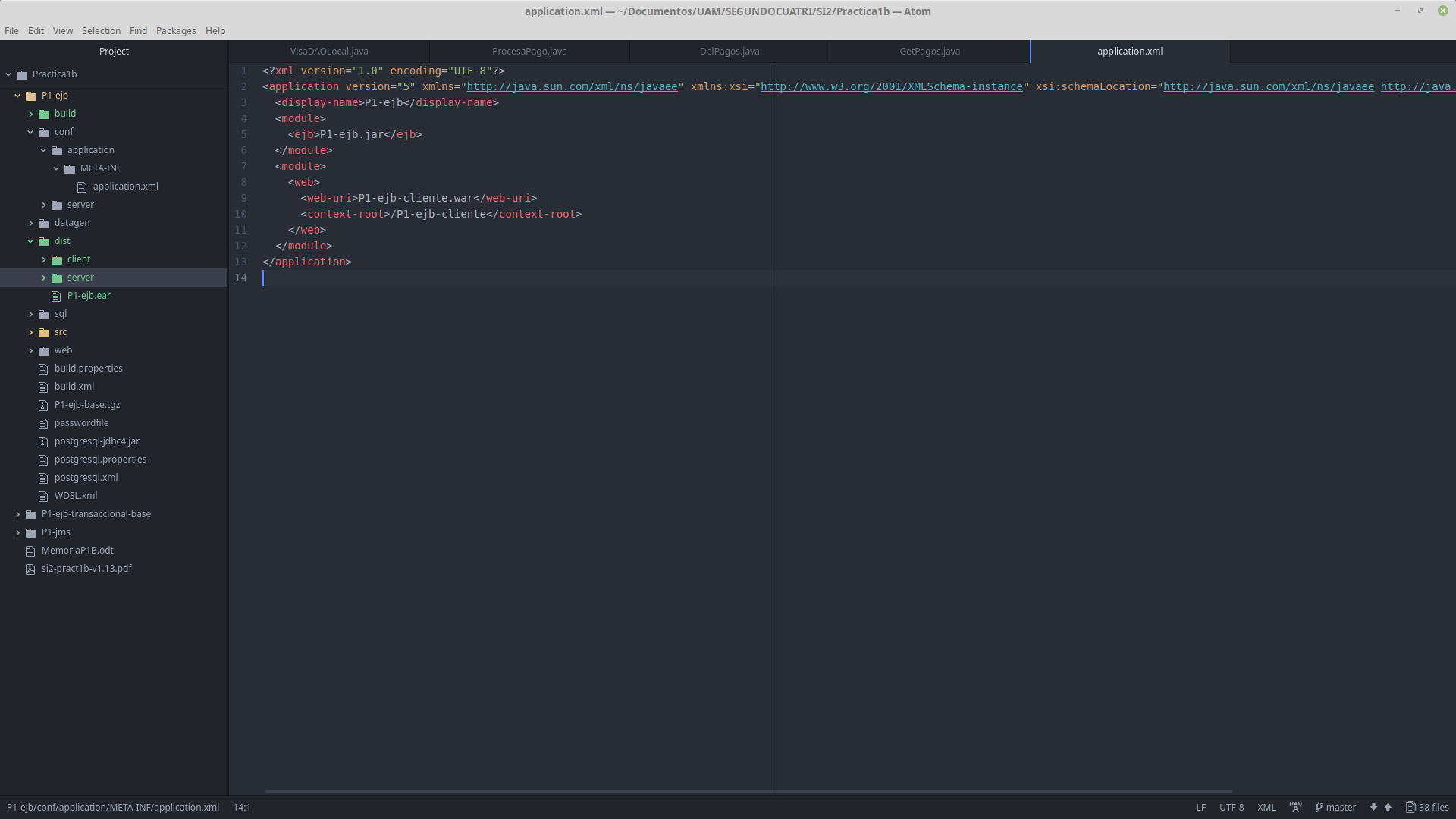
Todos los cambios se encuentran en el código que hemos modificado de la práctica. Como resumen, hemos seguido los pasos del enunciado: modificamos el servlet ProcesaPago.java añadiendo y eliminando unos imports, añadimos el objeto proxy al EJB local, eliminamos el webservice de la práctica anterior que estaba en un try catch y eliminamos las referencias a BindingProvider.

Además, hicimos los mismos cambios en los otros servlets que los necesitan, DelPagos.java y GetPagos.java, el cual hemos adaptado para que devuelva un array de PagoBean.

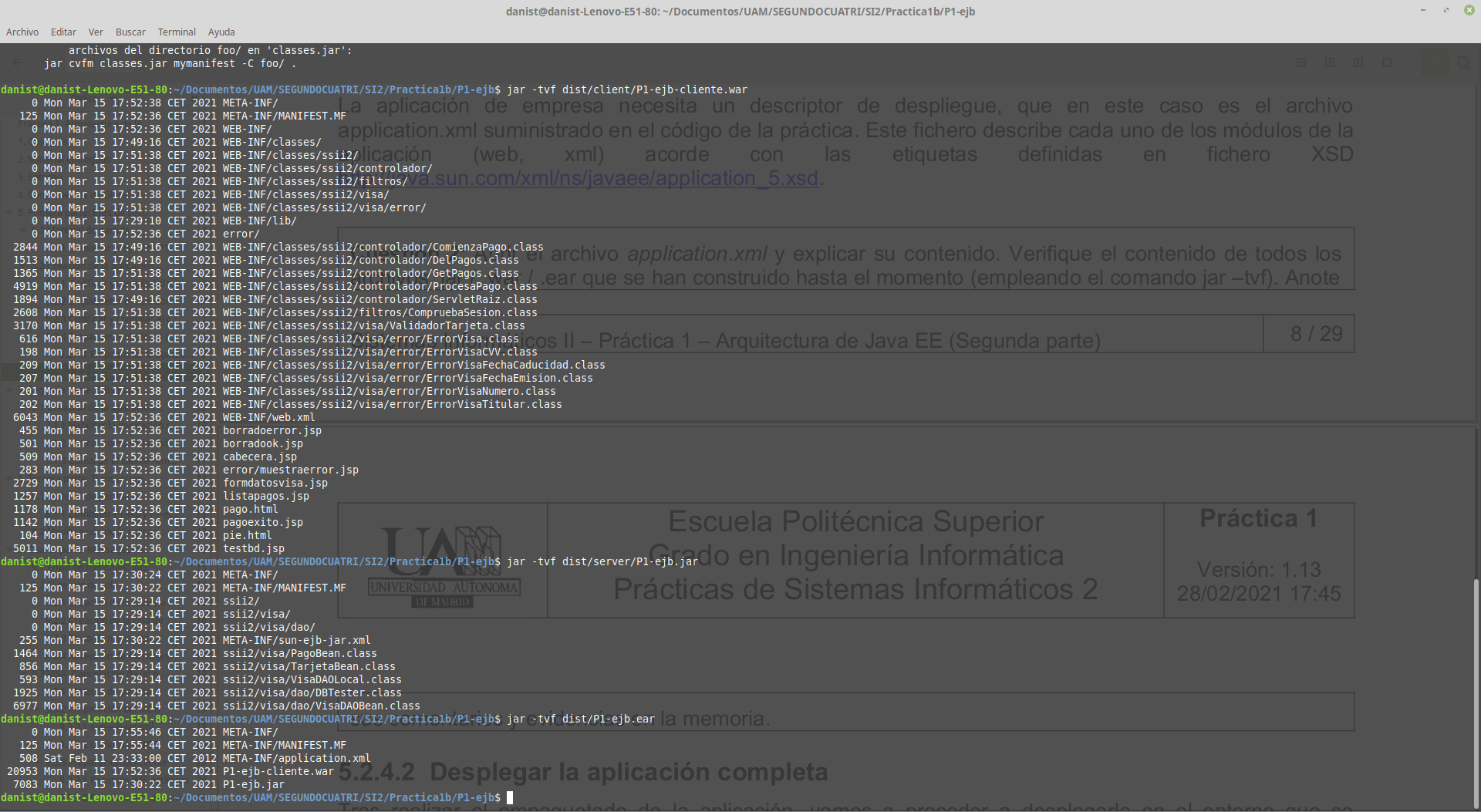
## Cuestión número 2:

**Editar el archivo application.xml y comprobar su contenido. Verifique el contenido de todos los archivos .jar /.war /.ear que se han construido hasta el momento (empleando el comando jar –tvf). Anote sus comentarios en la memoria.**

El archivo application.xml se encuentra en el directorio conf/application/META-INF, el cual hace referencia a P1-ejb.jar y a P1-ejb-cliente.war



Por un lado, el fichero .war contiene las páginas dinámicas y html, y además las clases de los servlets. El fichero .jar se genera cuando empaquetamos el servidor y contiene las clases del mismo y el descriptor del EJB. Finalmente, el fichero .ear contiene los ficheros anteriores y la descripción del application.xml, generándose al empaquetar la aplicación. Se adjunta la captura de dichos ficheros:



***Ejercicio 3:***

**Preparar los PCs con el esquema descrito y realizar el despliegue de la aplicación:**

** Editar el archivo build.properties para que las propiedades as.host.client y as.host.server contengan la dirección IP del servidor de aplicaciones. Indica qué valores y porqué son esos valores.**

** Editar el archivo postgresql.properties para la propiedad db.client.host y db.host contengan las direcciones IP adecuadas para que el servidor de aplicaciones se conecte al postgresql, ambos estando en servidores diferentes. Indica qué valores y porqué son esos valores.**

**Desplegar la aplicación de empresa**

Ahora las IPS son 10.2.1.2 para el cliente y servidor y 10.2.1.1 para la base de datos, luego modificamos el build.properties y postgre.sql como primer paso:

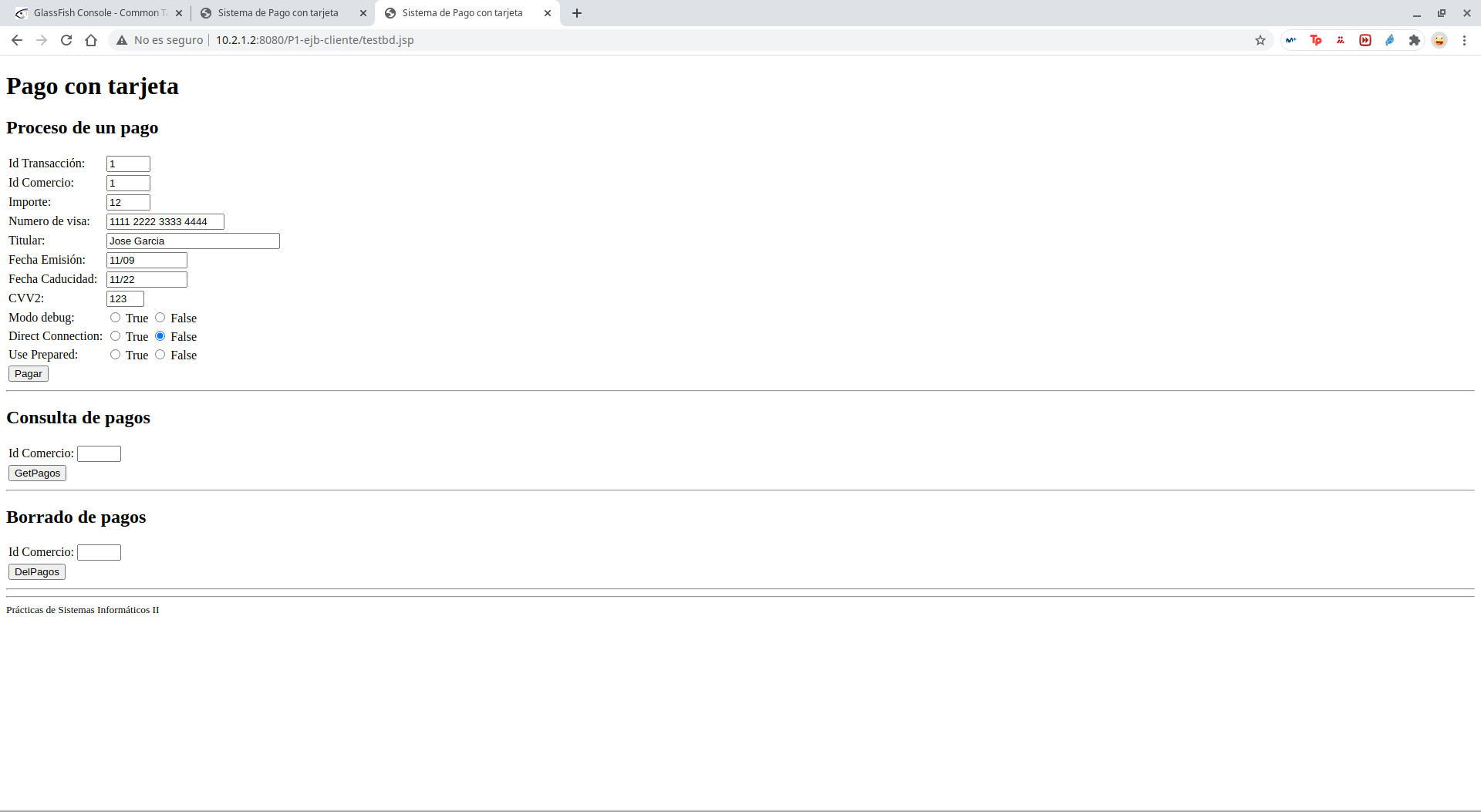
* as.host.client = 10.2.1.2
* as.host.server = 10.2.1.2
* db.host = 10.2.1.1
* db.client.host = 10.2.1.2

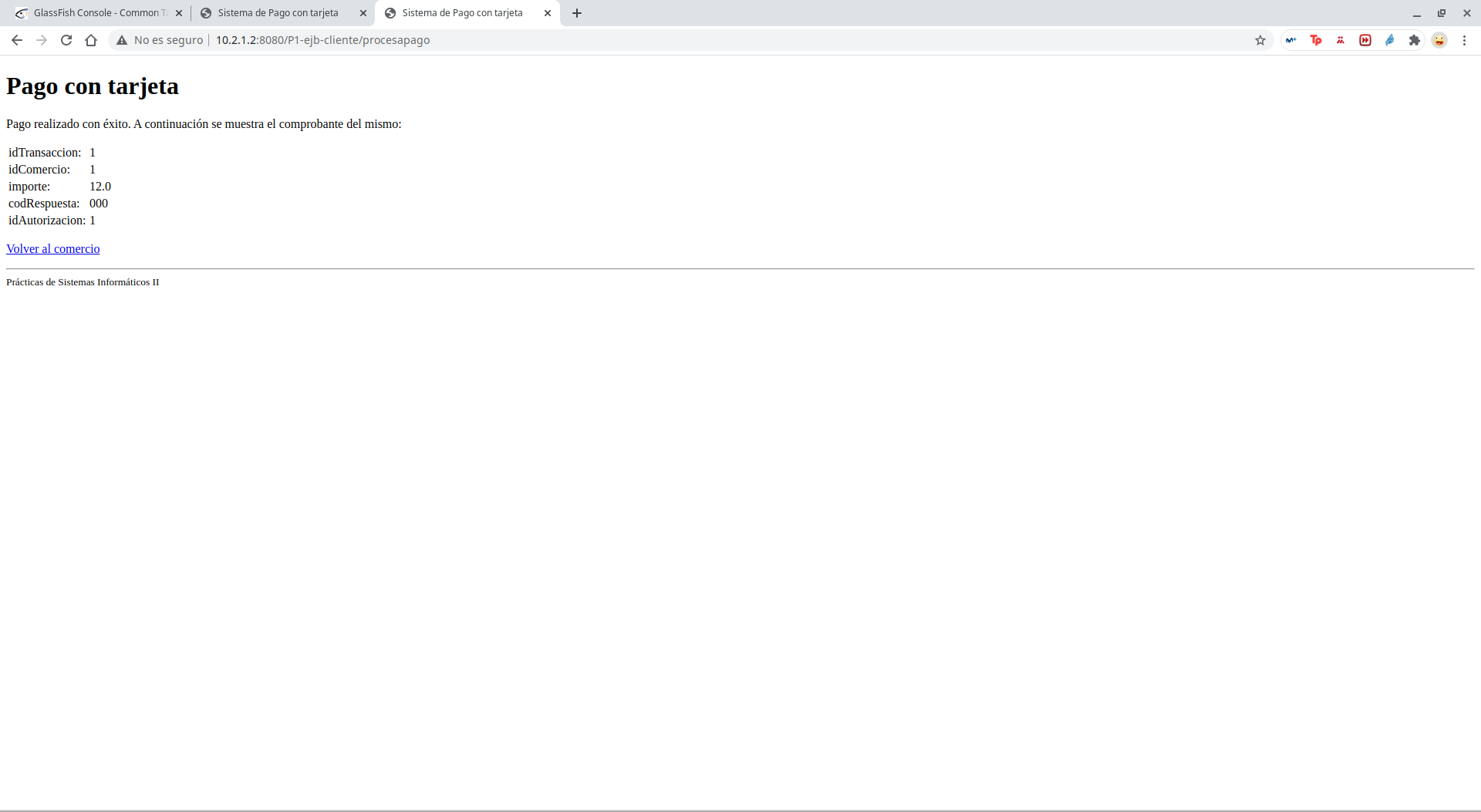
Desplegamos la aplicación con “ant desplegar” correctamente y pasamos al ejercicio siguiente para demostrar que verdaderamente funciona.

***Ejercicio número 4:***

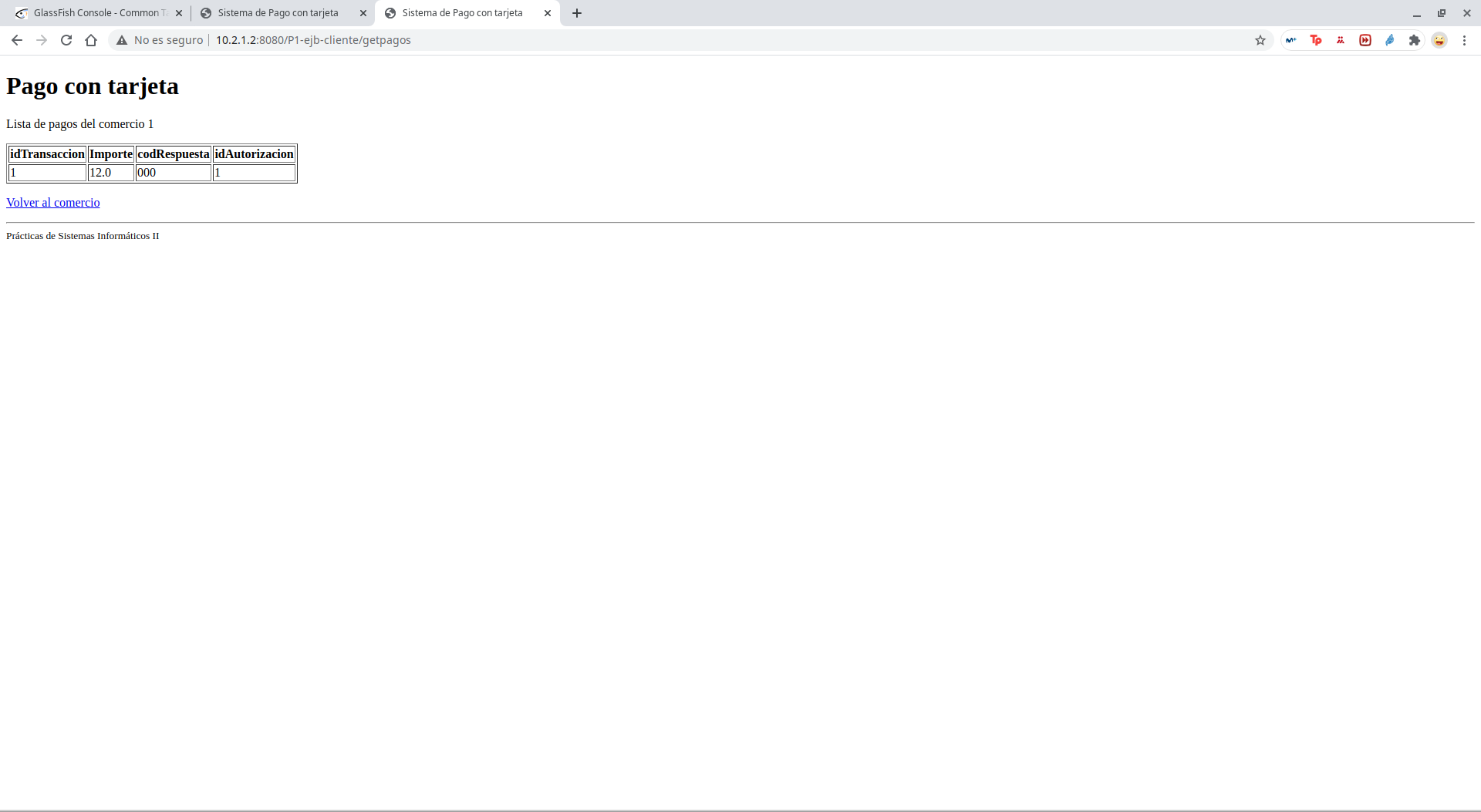
**Comprobar el correcto funcionamiento de la aplicación mediante llamadas directas a través de las páginas pago.html y testbd.jsp (sin directconnection). Realice un pago. Lístelo. Elimínelo. Téngase en cuenta que la aplicación se habrá desplegado bajo la ruta /P1-ejbcliente.**

Primero realizamos un pago:

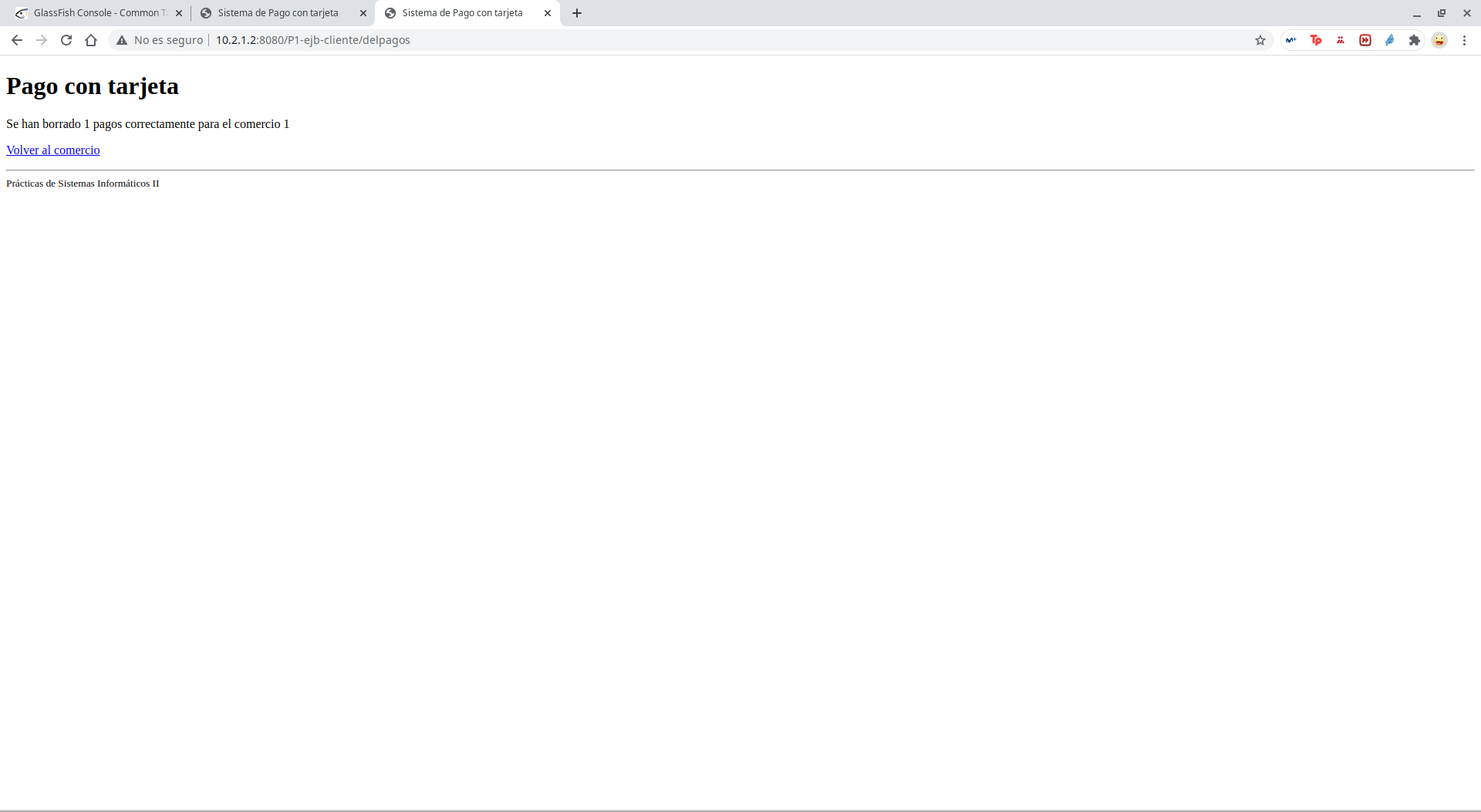




Luego comprobamos que se haya realizado accediendo a la lista de pagos:



Por último, eliminamos el pago:



***Ejercicio 5:***

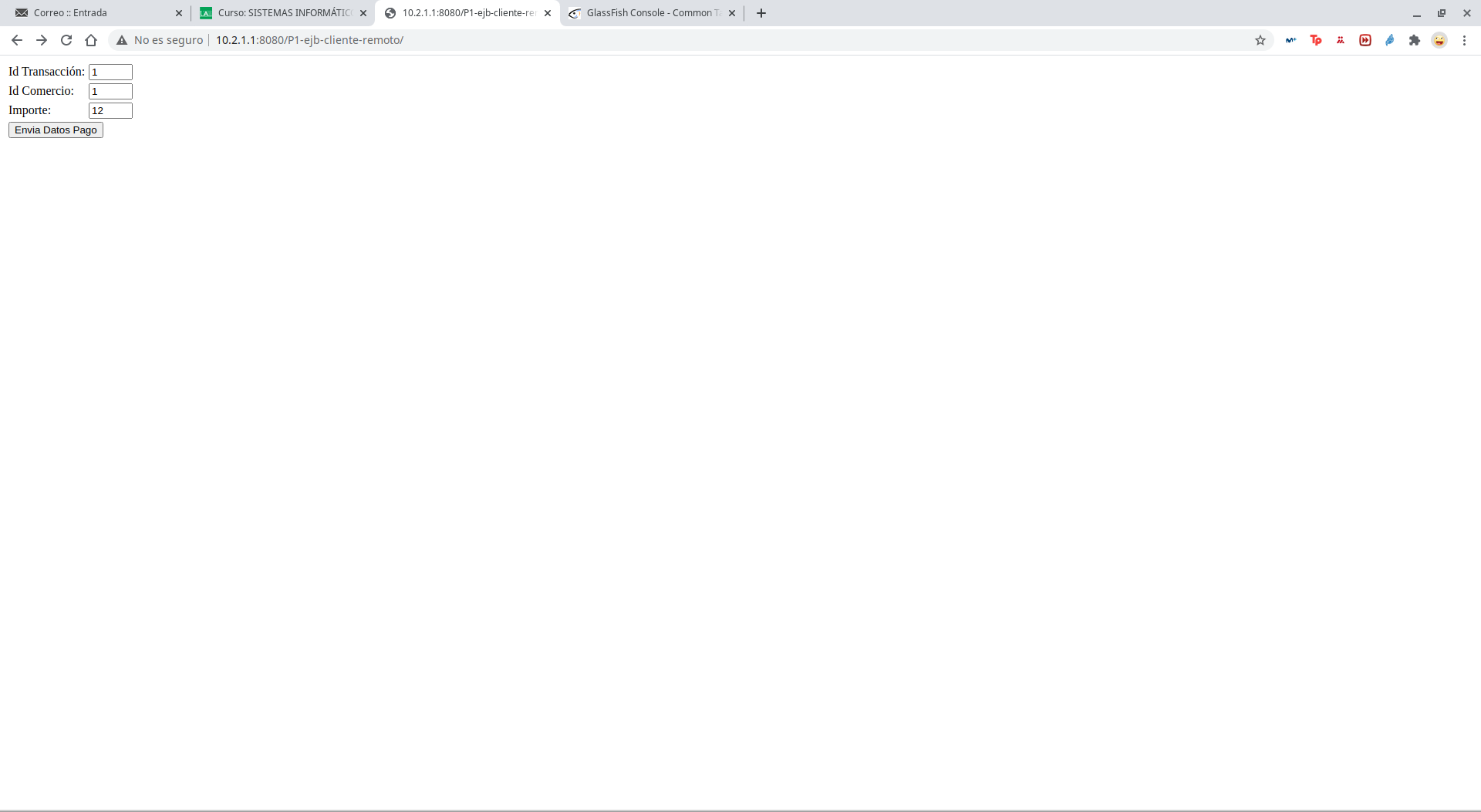
**Realizar los cambios indicados en P1-ejb-servidor-remoto y preparar los PCs con el esquema de máquinas virtuales indicado. Compilar, empaquetar y desplegar de nuevo la aplicación P1- ejb como servidor de EJB remotos de forma similar a la realizada en el Ejercicio 3 con la Figura 2 como entorno de despliegue. Esta aplicación tendrá que desplegarse en la máquina virtual del PC2. Se recomienda replegar la aplicación anterior (EJB local) antes de desplegar ésta. Incluye en la memoria cada fragmento de código donde se han ido añadiendo las modificaciones solicitadas, así como detallando los pasos realizados.**

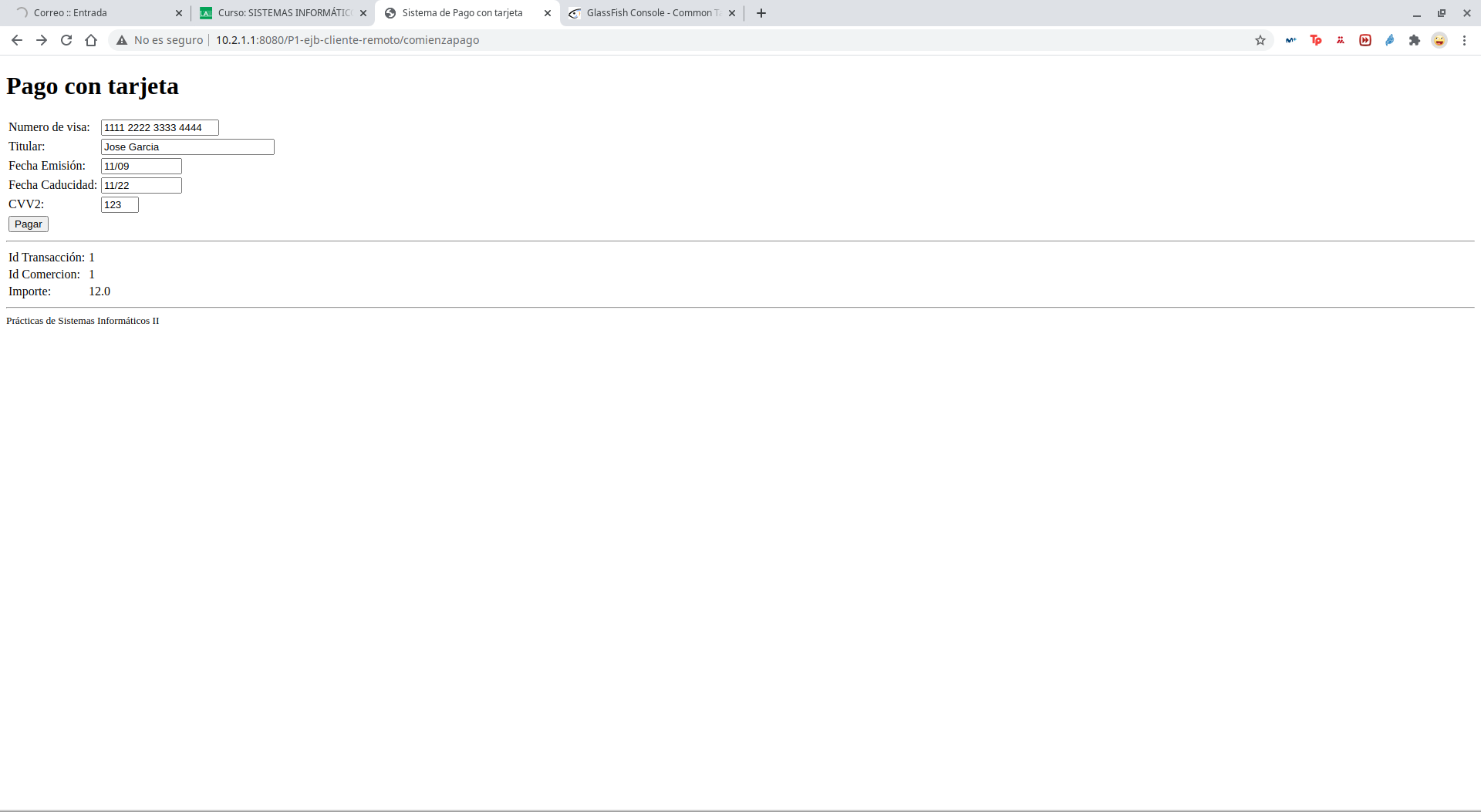
Hemos creado la interfaz VisaDAORemote a partir de VisaDAOLocal pero poniéndole la etiqueta @Remote. También implementamos VisaDAOBean como interfaz remota y hemos hecho que PagoBean y TarjetaBean sean serializables. Volvemos a desplegar la aplicación como en el ejercicio anterior y hemos visto que funciona correctamente.

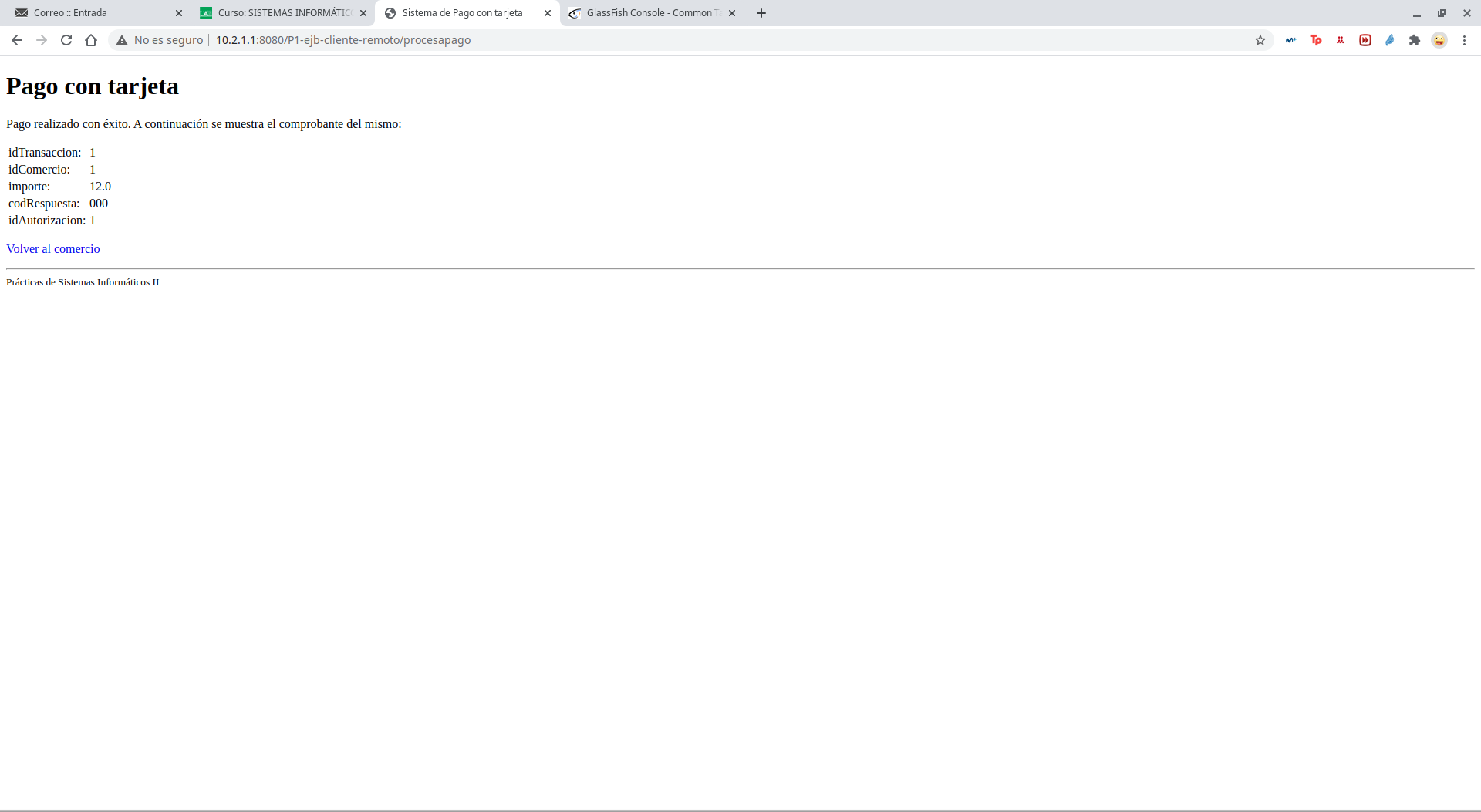
***Ejercicio 6:***

**Realizar los cambios comentados en la aplicación P1-base para convertirla en P1-EJBclienteremoto y compilar, empaquetar y desplegar de nuevo la aplicación en otra máquina virtual distinta a la de la aplicación servidor, es decir, esta aplicación cliente estará desplegada en la MV del PC1 tal y como se muestra en la Figura 2. Conectarse a la aplicación cliente y probar a realizar un pago. Comprobar los resultados e incluir en la memoria evidencias de que el pago ha sido realizado de forma correcta.**

Tras realizar todos los cambios del enunciado, volvemos a desplegar la aplicación para el cliente remoto en la dirección IP 10.2.1.1 y pasamos a realizar un pago y comprobar que verdaderamente se ha realizado:





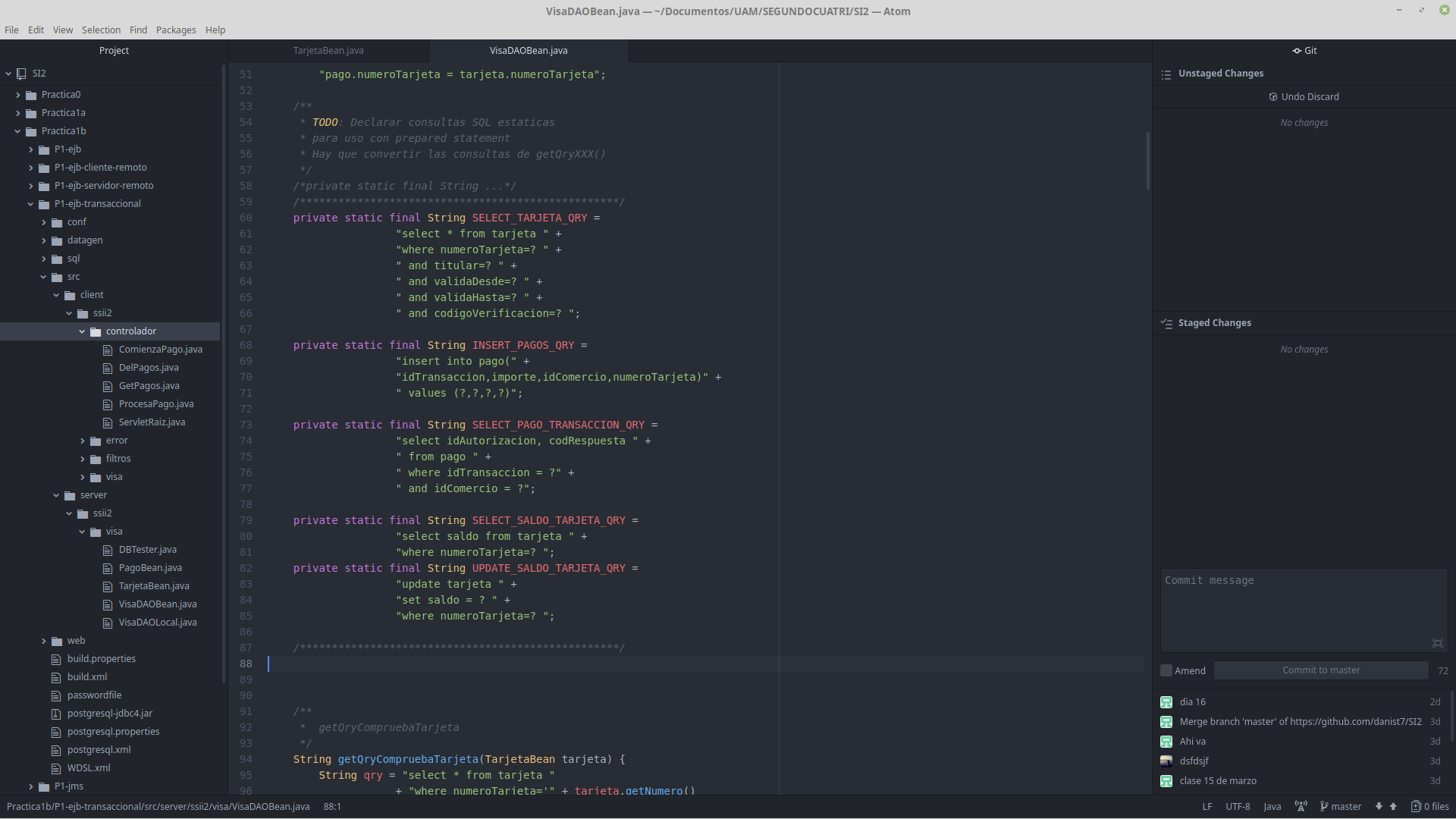


***Ejercicio 7:***

**Modificar la aplicación VISA para soportar el campo saldo.**

Hemos seguido los pasos del enunciado:

* Añadimos el atributo saldo a TarjetaBean.java como double, así como sus métodos.
* Declaramos dos statements para hacer un select y un update del saldo en la base de datos.



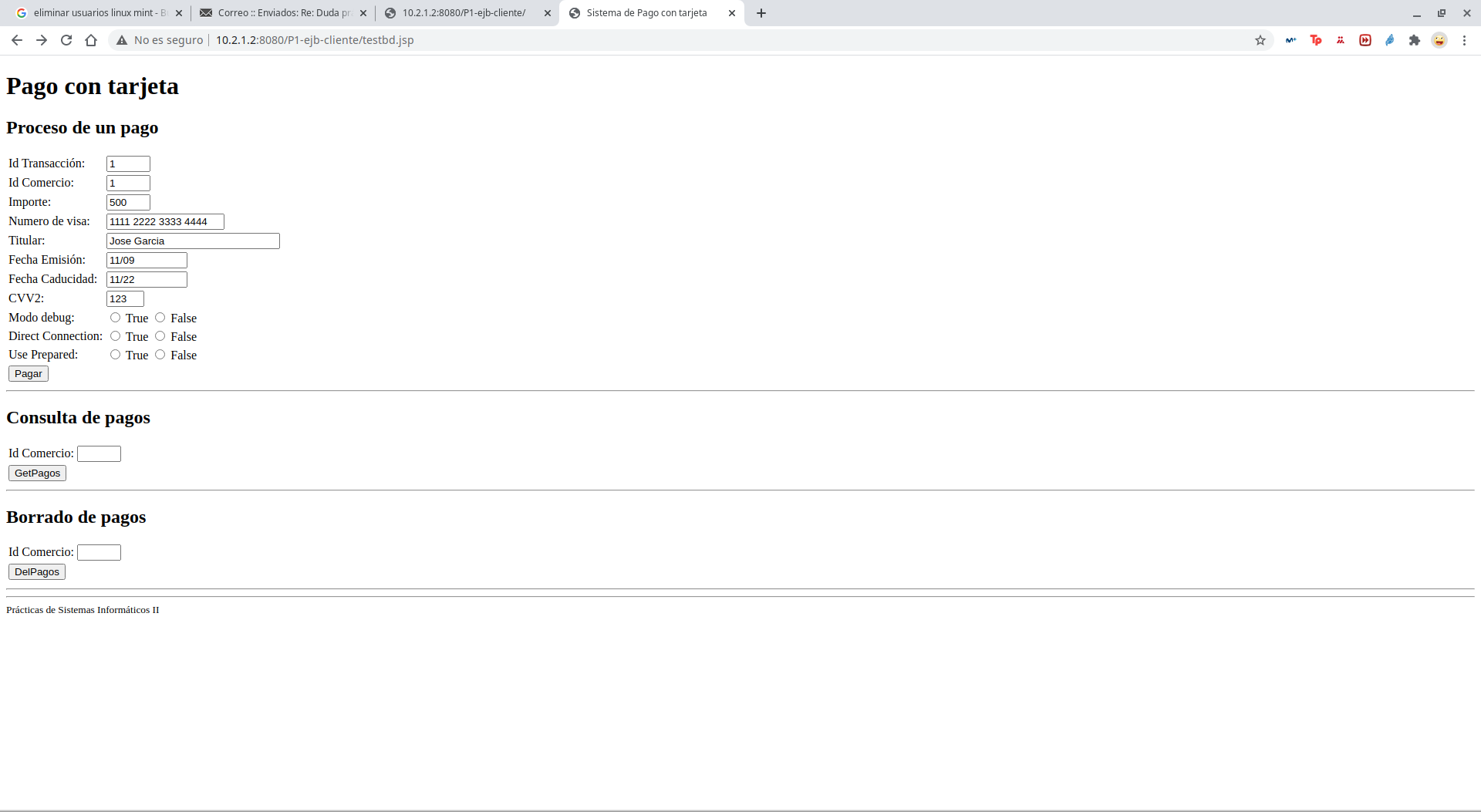
* Modificamos el método realizaPago y el servlet ProcesaPago para adaptarlos al saldo.

***Ejercicio 8:***

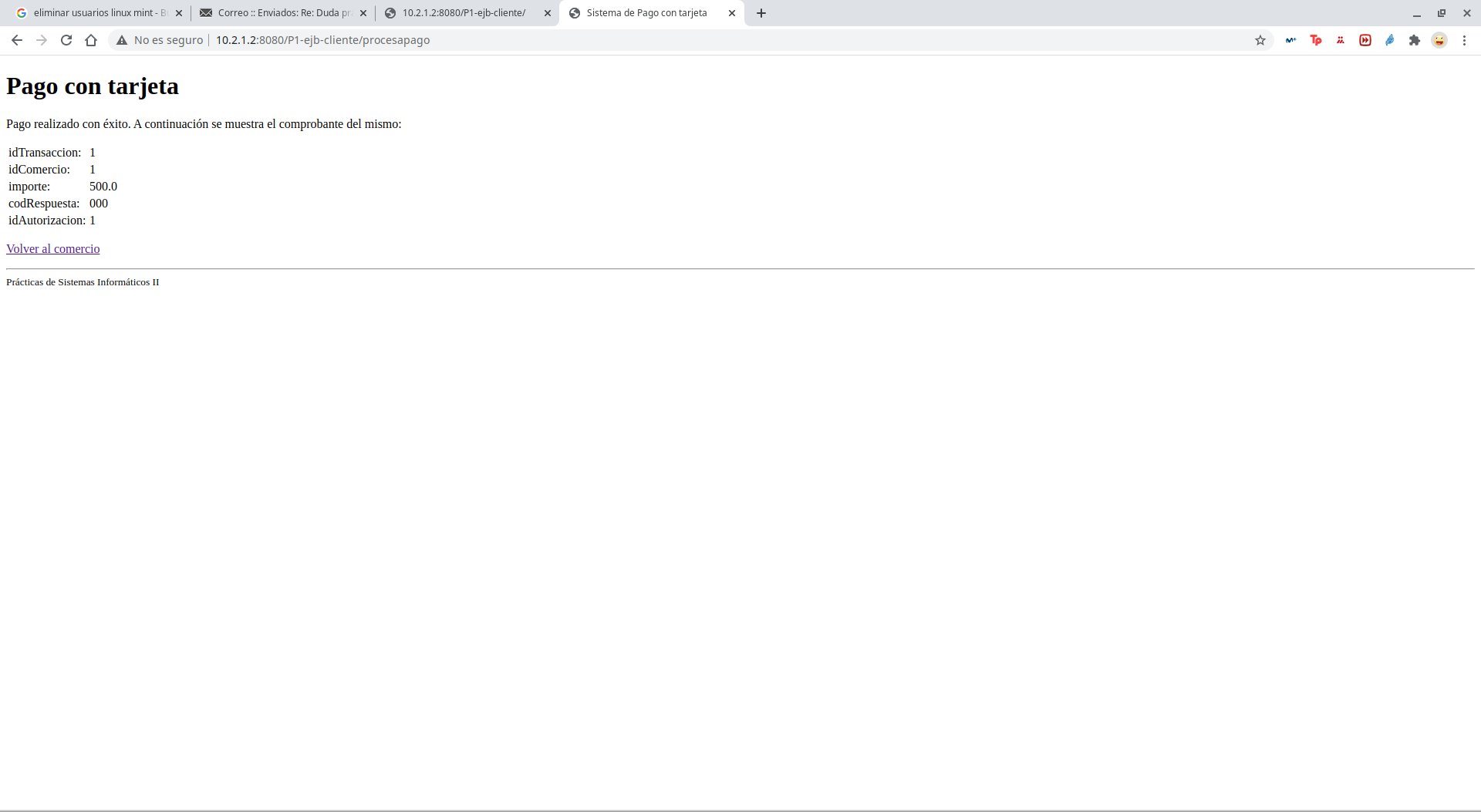
**Desplegar y probar la nueva aplicación creada.**

Todos los saldos están inicializados a 1000. De modo que a continuación podemos observar cómo ha disminuido el saldo al realizar el pago.

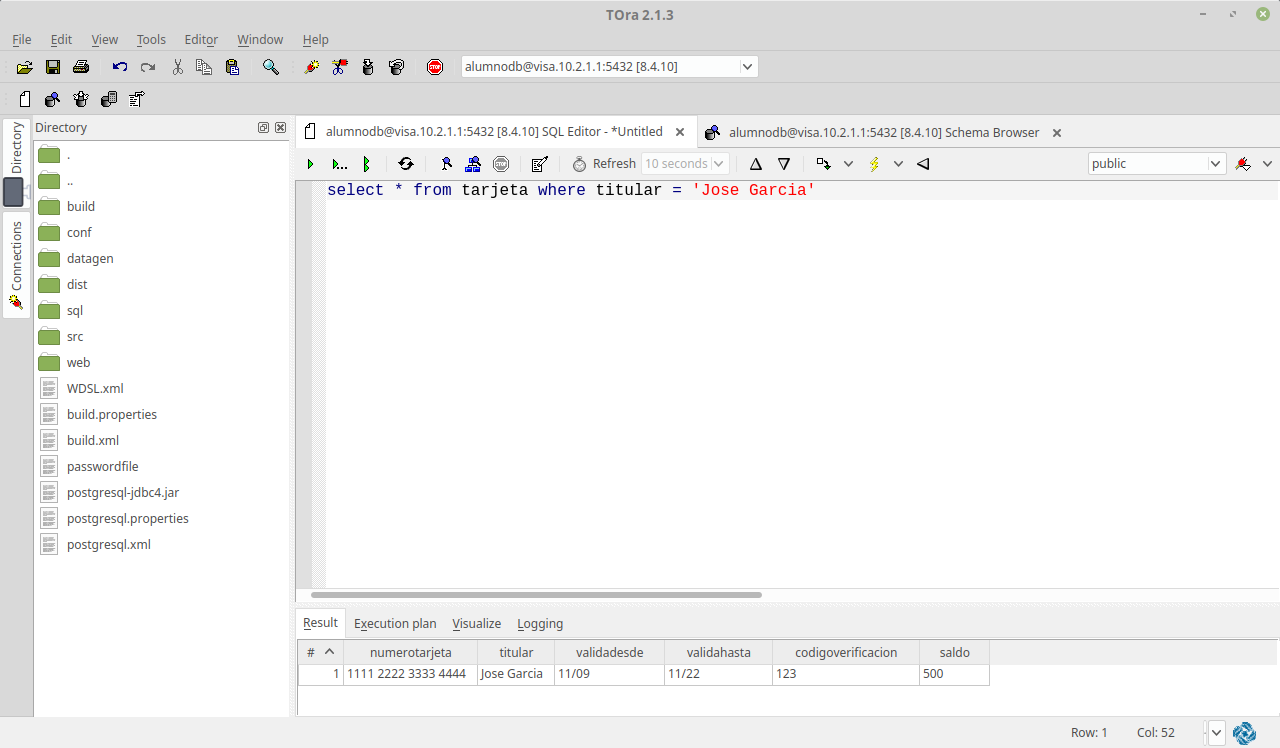
1. Realizamos un pago: le quitamos 500 euros a Jose García.



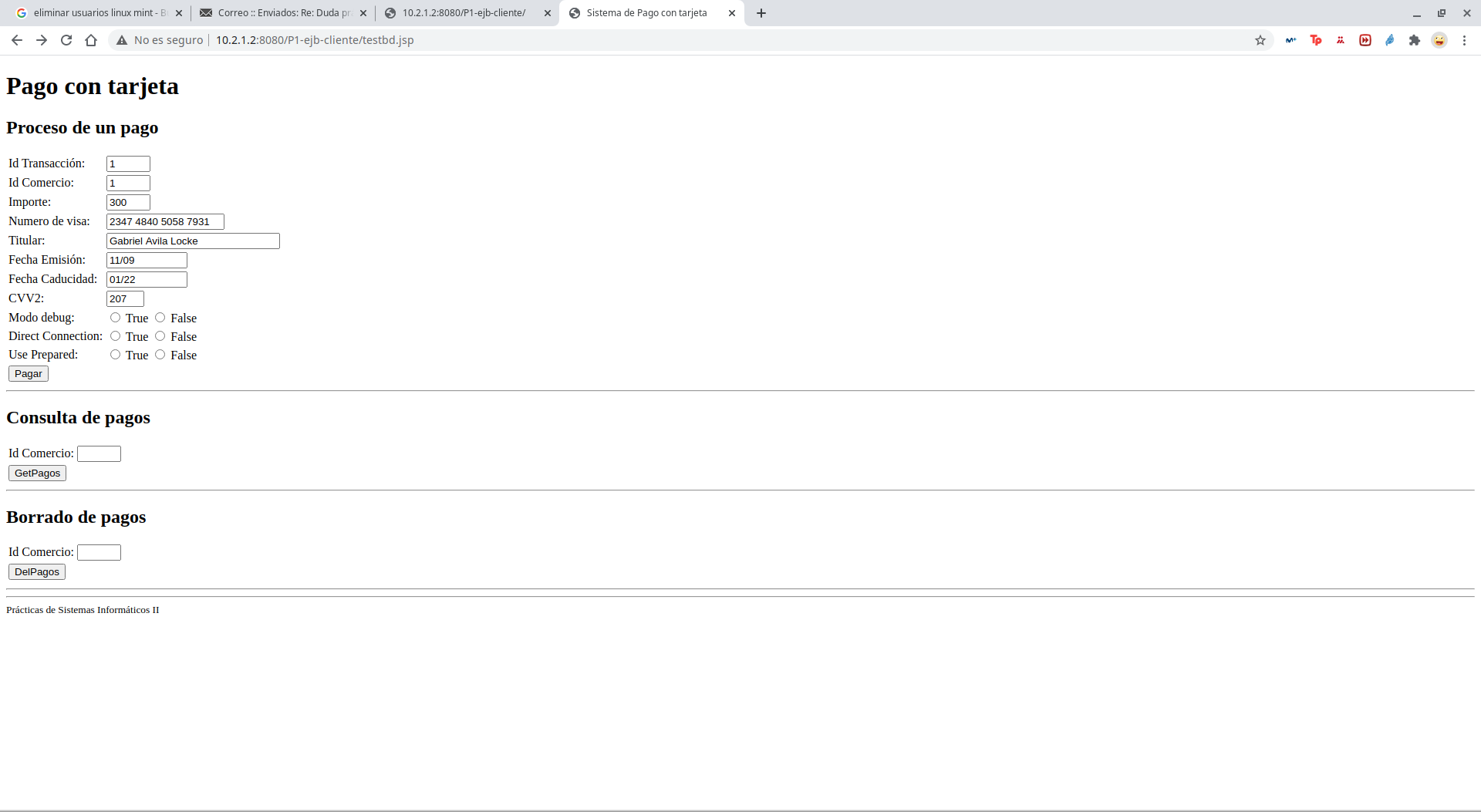
1. Pago realizado con éxito:



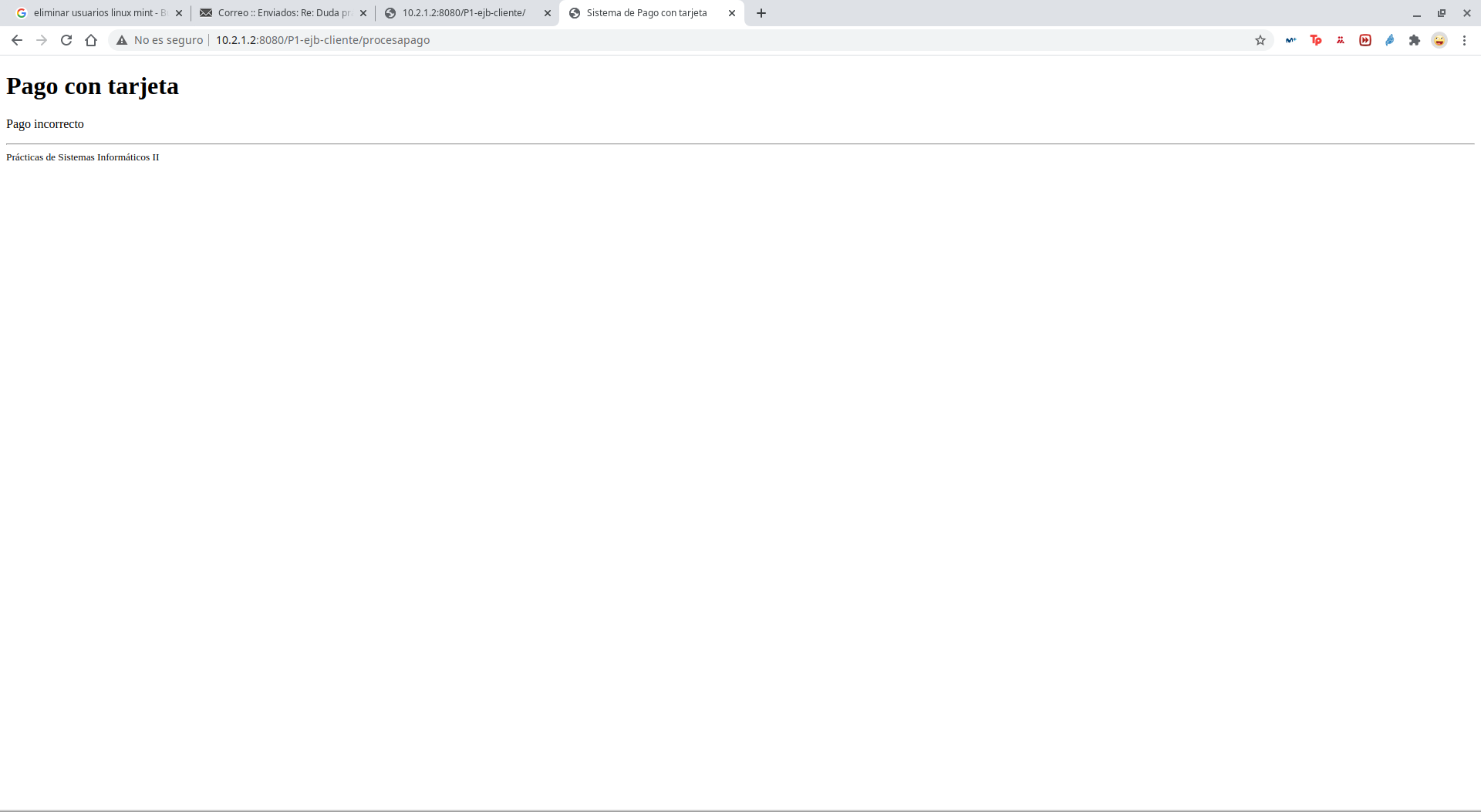
1. Miramos la base de datos y comprobamos que de 1000 euros, la cuenta de Jose García ha bajado 500 euros:



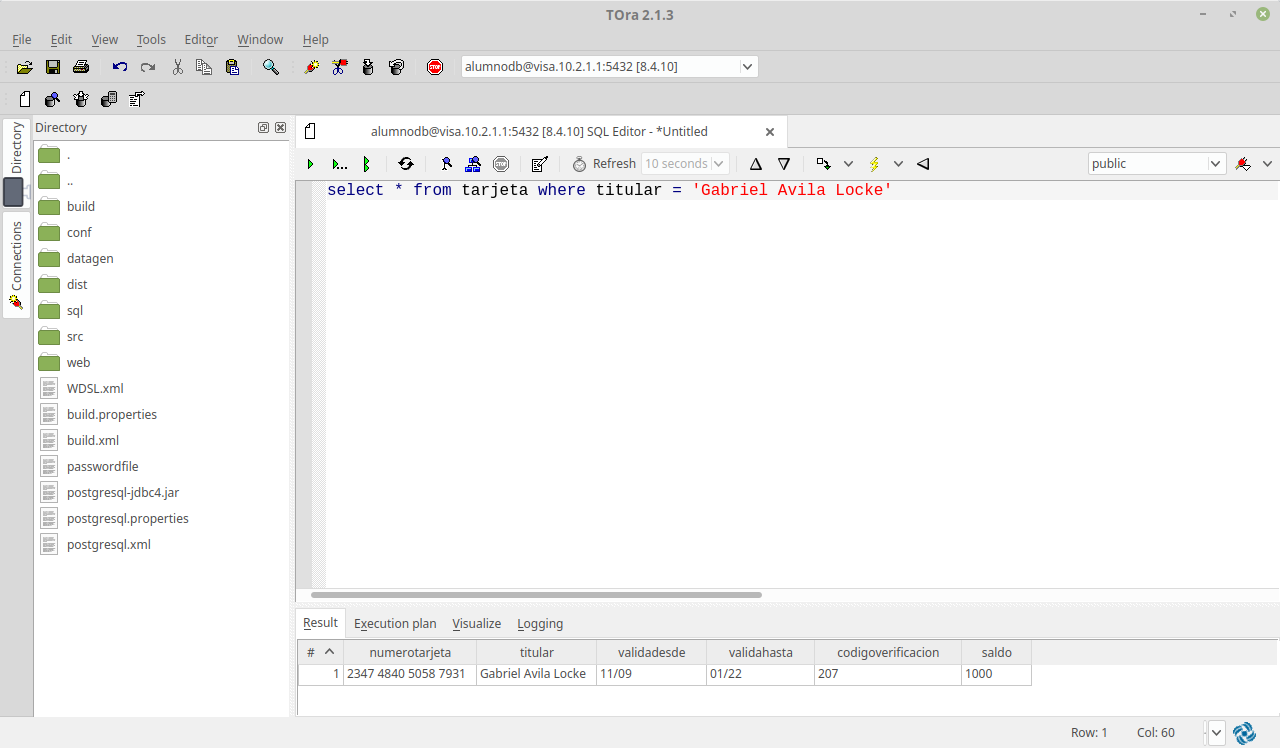
1. Realizamos una operación con identificador de transacción y de comercio duplicados para luego ver que ha sido erróneo. Ahora utilizamos la cuenta de Gabriel Avila.



1. Comprobamos que el pago es erróneo:

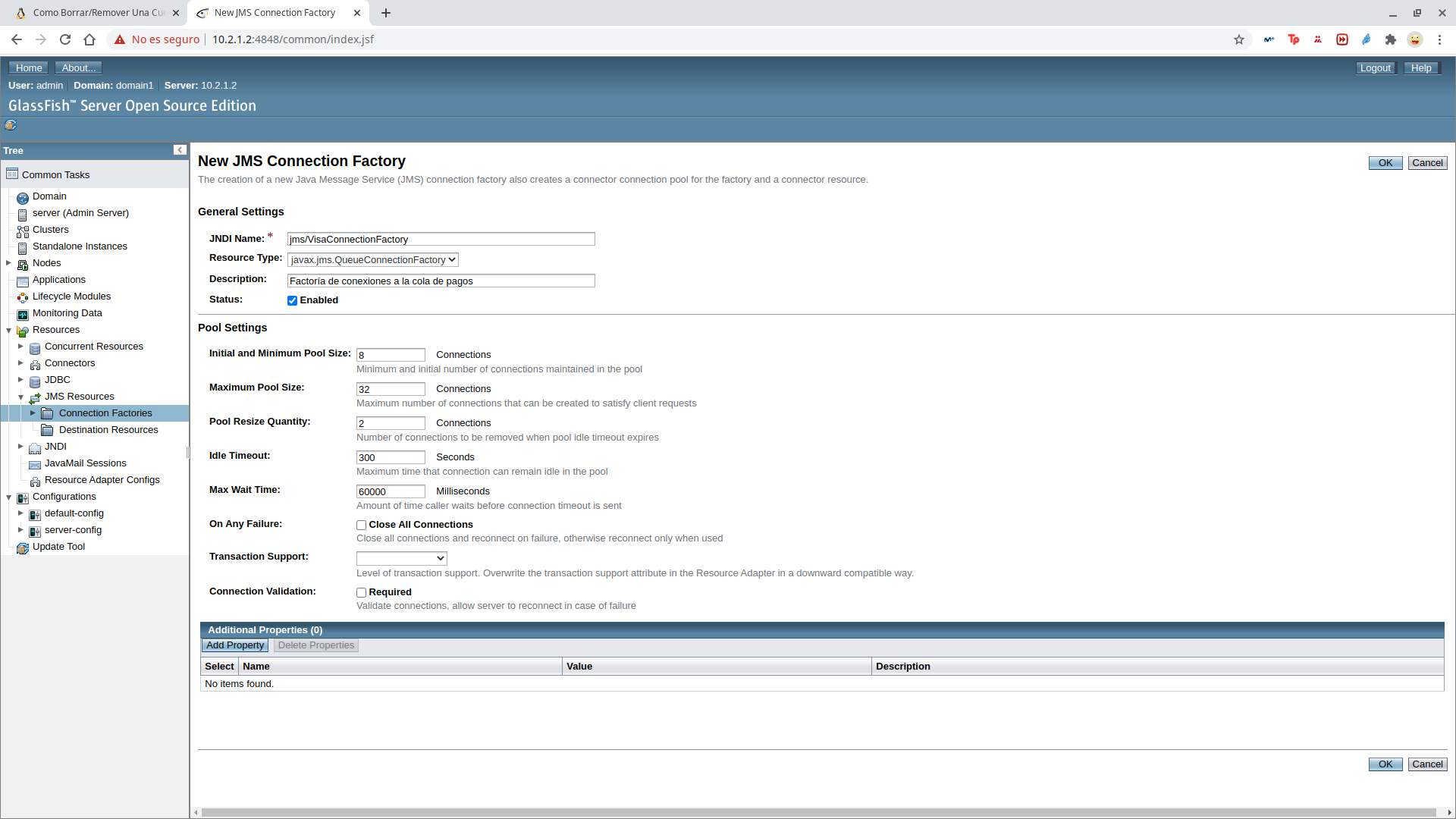


1. Vemos en la base de datos que su salgo no ha variado pues sigue a mil.



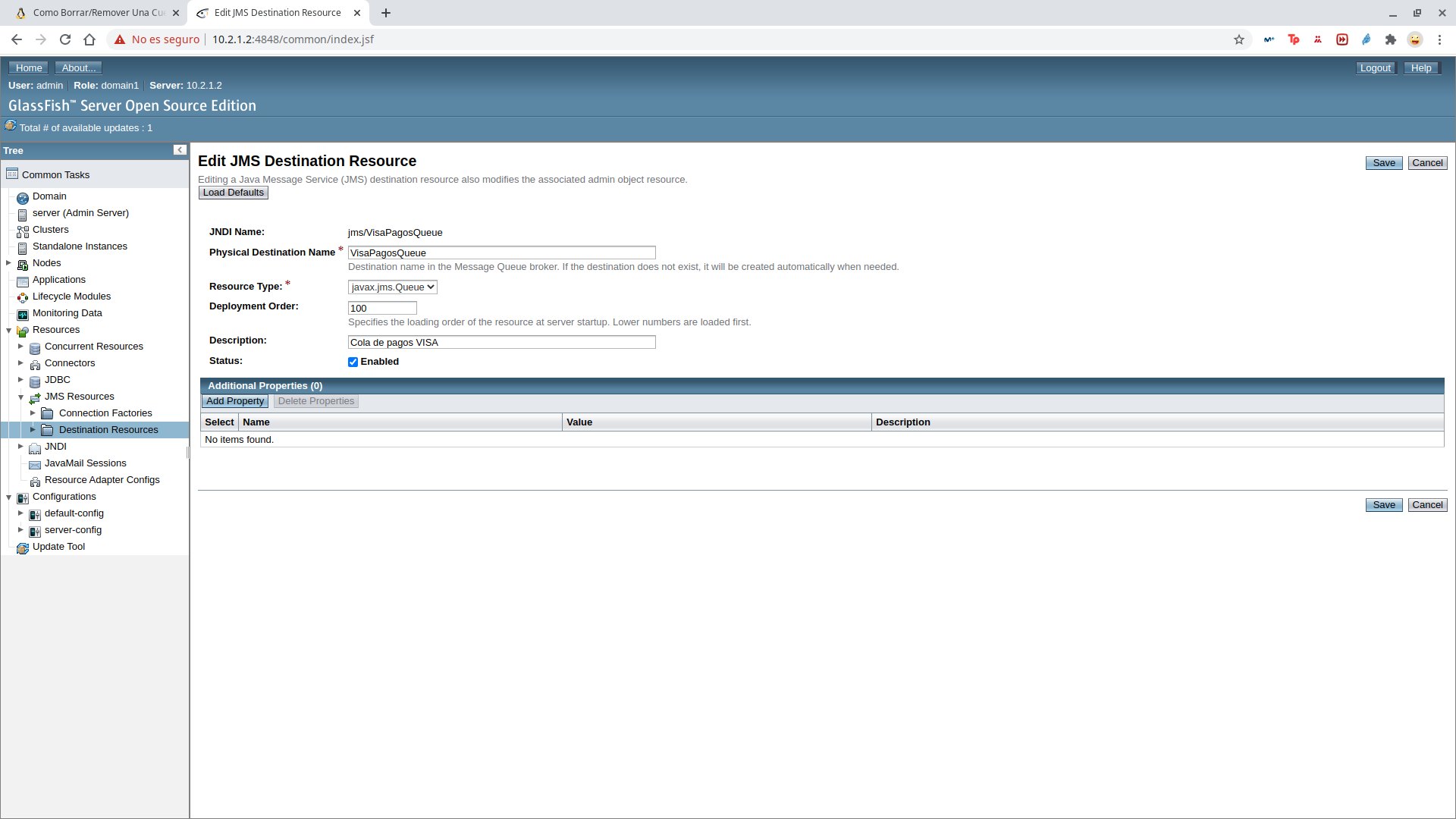
***Ejercicio 9:***

**En la máquina virtual donde se encuentra el servidor de aplicaciones (10.X.Y.2), declare manualmente la factoría de conexiones empleando la consola de administración, tal y como se adjunta en la Figura 4.**



***Ejercicio 10:***

**En la máquina virtual donde se encuentra el servidor de aplicaciones (10.X.Y.2), declare manualmente la conexión empleando la consola de administración, tal y como se adjunta en la Figura 5.**



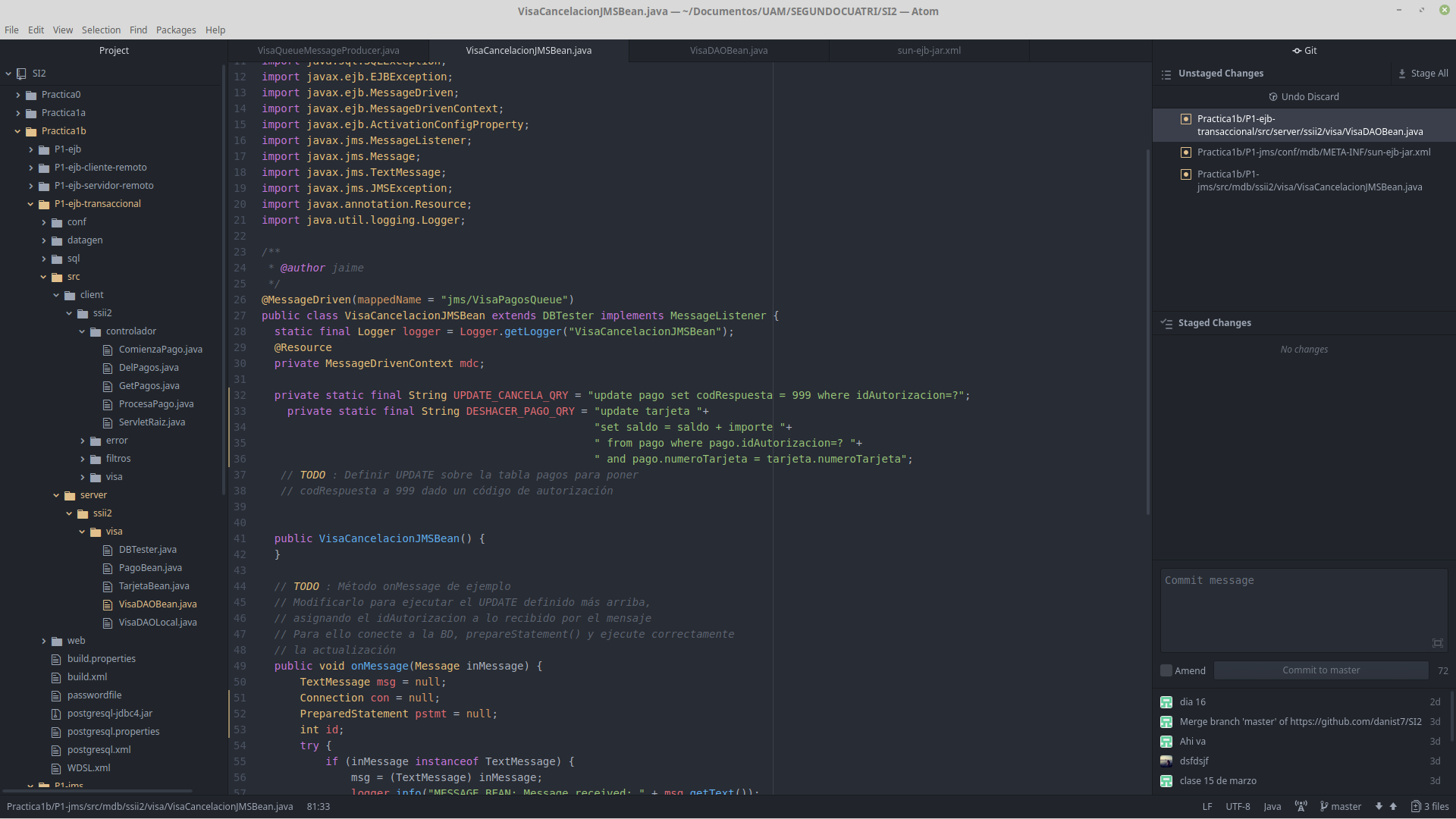
***Ejercicio 11:***

**Modifique el fichero sun-ejb-jar.xml para que el MDB conecte adecuadamente a su connection factory. Incluya en la clase VisaCancelacionJMSBean: (…)**

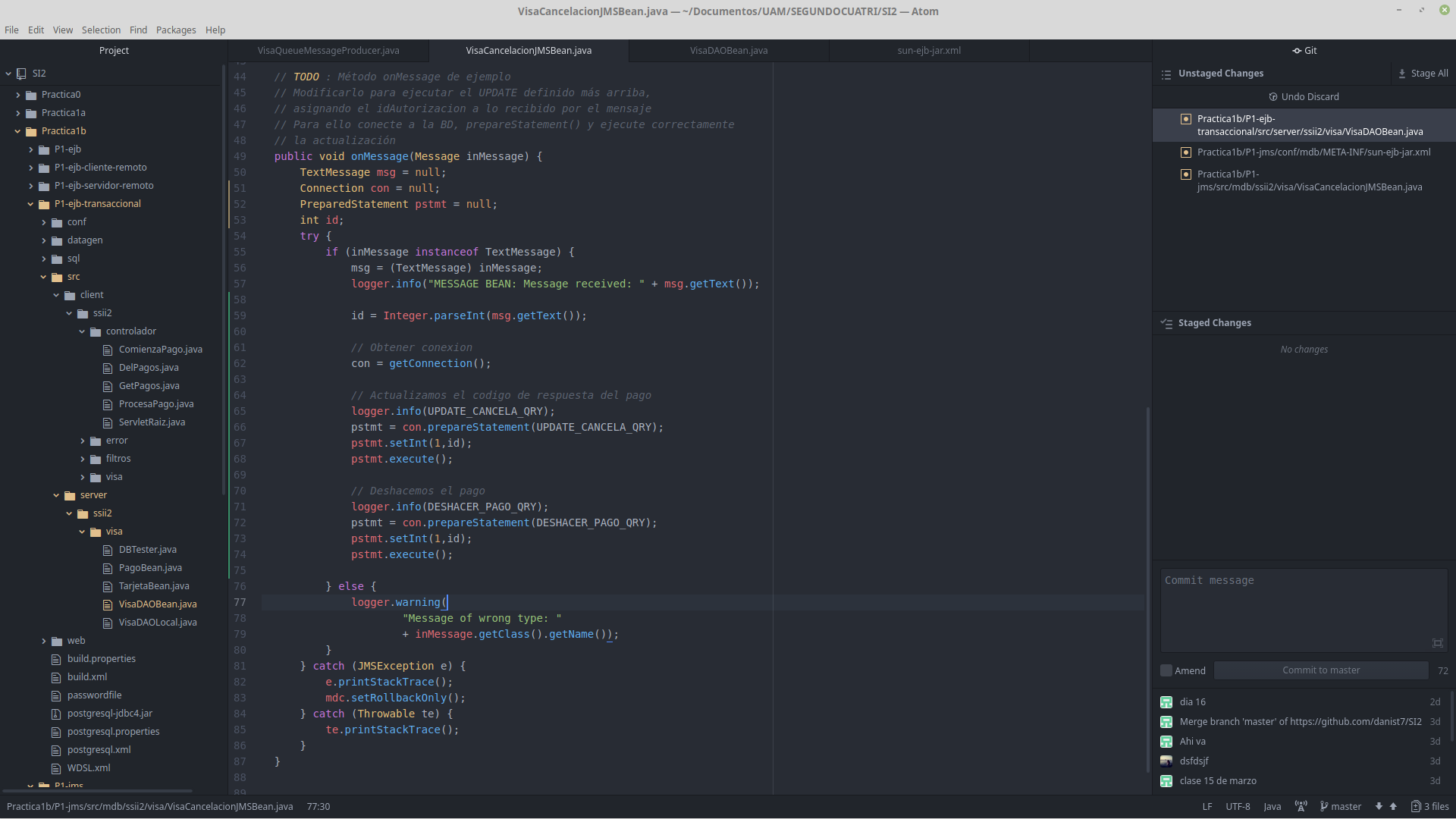
Modificamos el fichero sun-ejb-jar.xml: para ello simplemente añadimos la siguiente sentencia:

<jndi-name>jms/VisaConnectionFactory</jndi-name>

Añadimos las sentencias SQL en VisaCancelacionJMSBean:



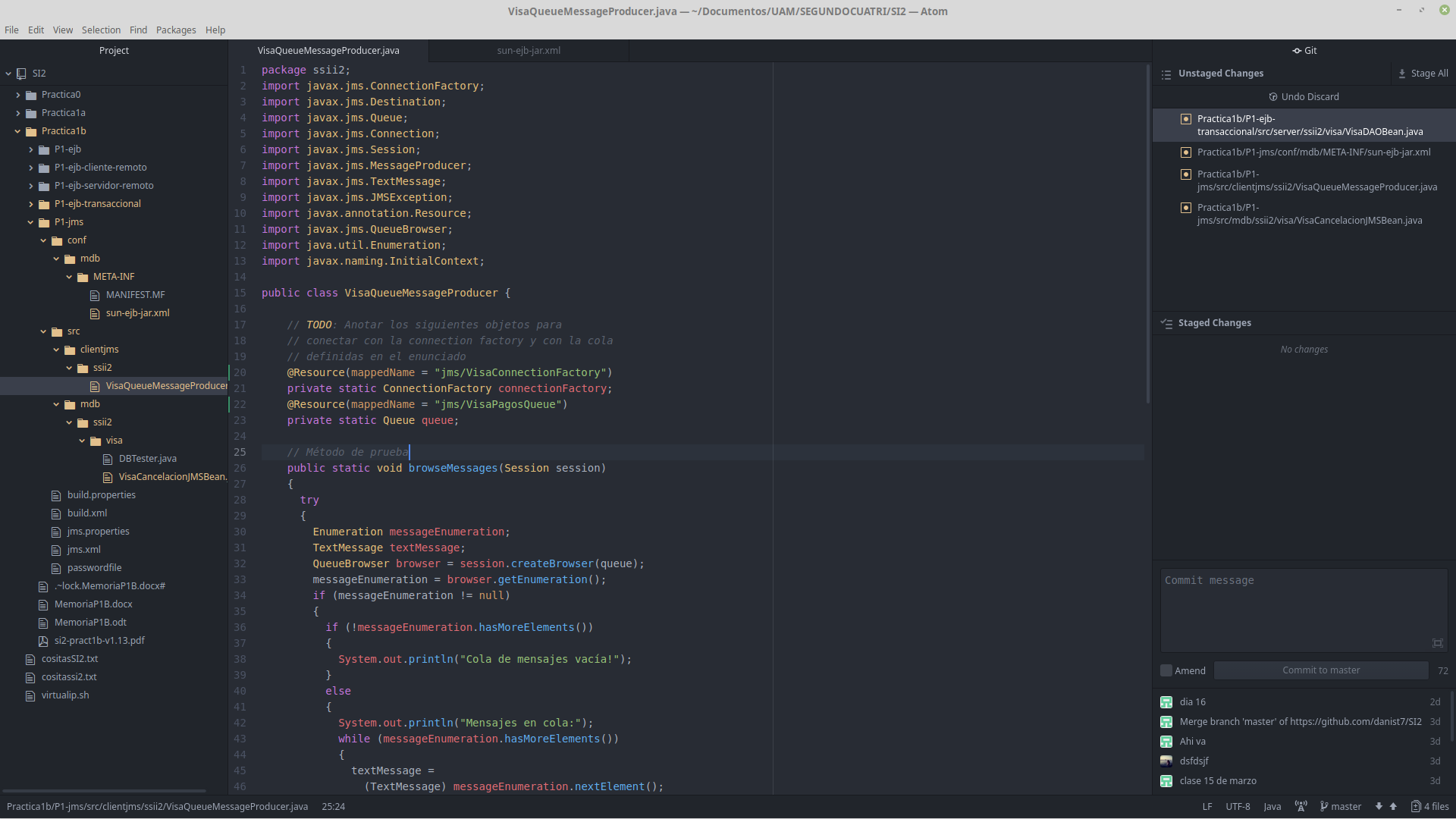
Implementamos el método onMessage() para implementar ambas actualizaciones SQL y con el control de errores:



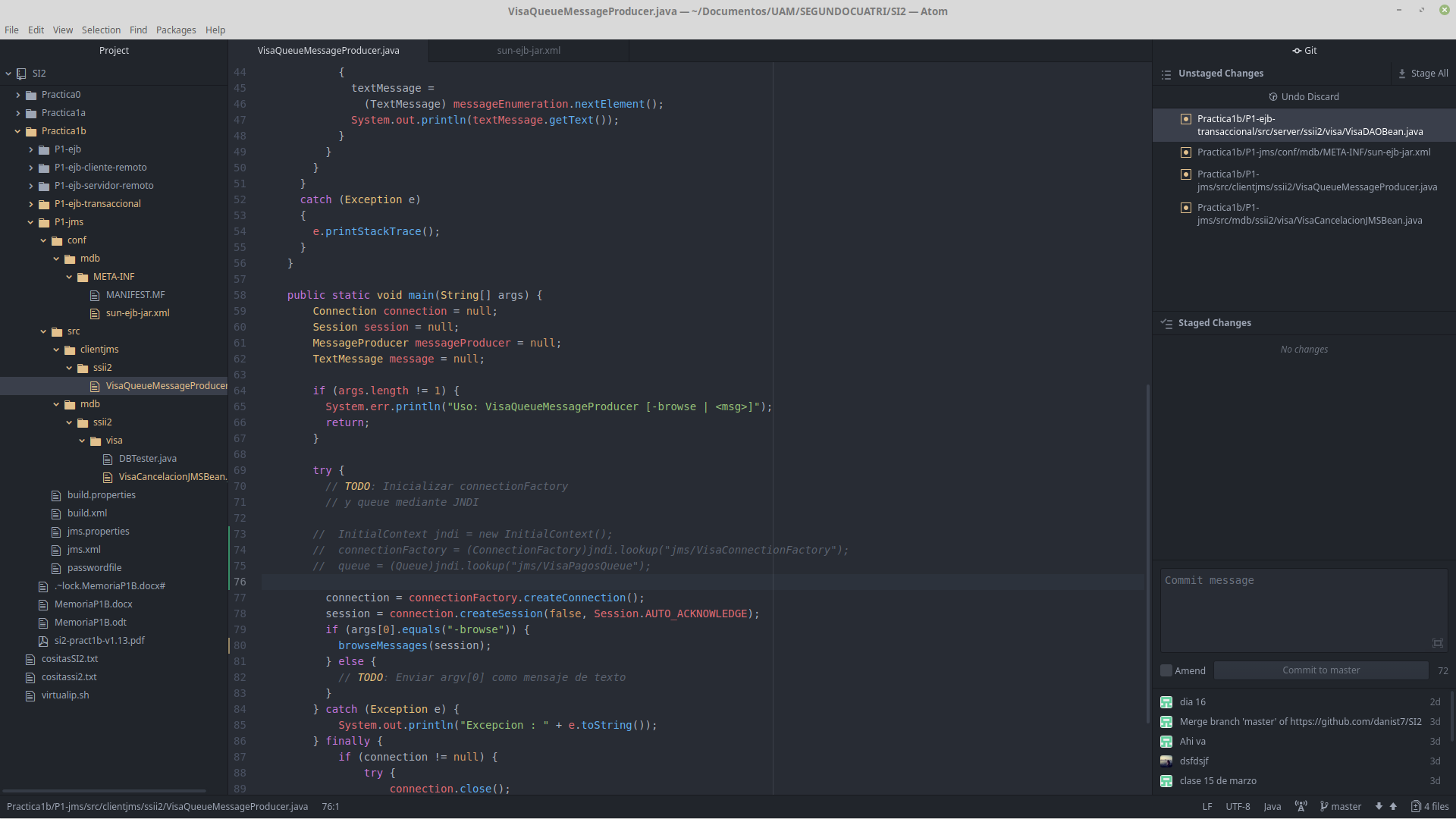
***Ejercicio 12:***

**Implemente ambos métodos en el cliente proporcionado. Deje comentado el método de acceso por JNDI. Indique en la memoria de prácticas qué ventajas podrían tener uno u otro método.**

Añadimos los Resources en las variables queue y connectionFactory:



Y luego hacemos la búsqueda mediante JDNI, lo dejamos también comentado en el código. En este método JDNI damos transparencia de ubicación ya que no es necesario saber la ubicación de los recursos.



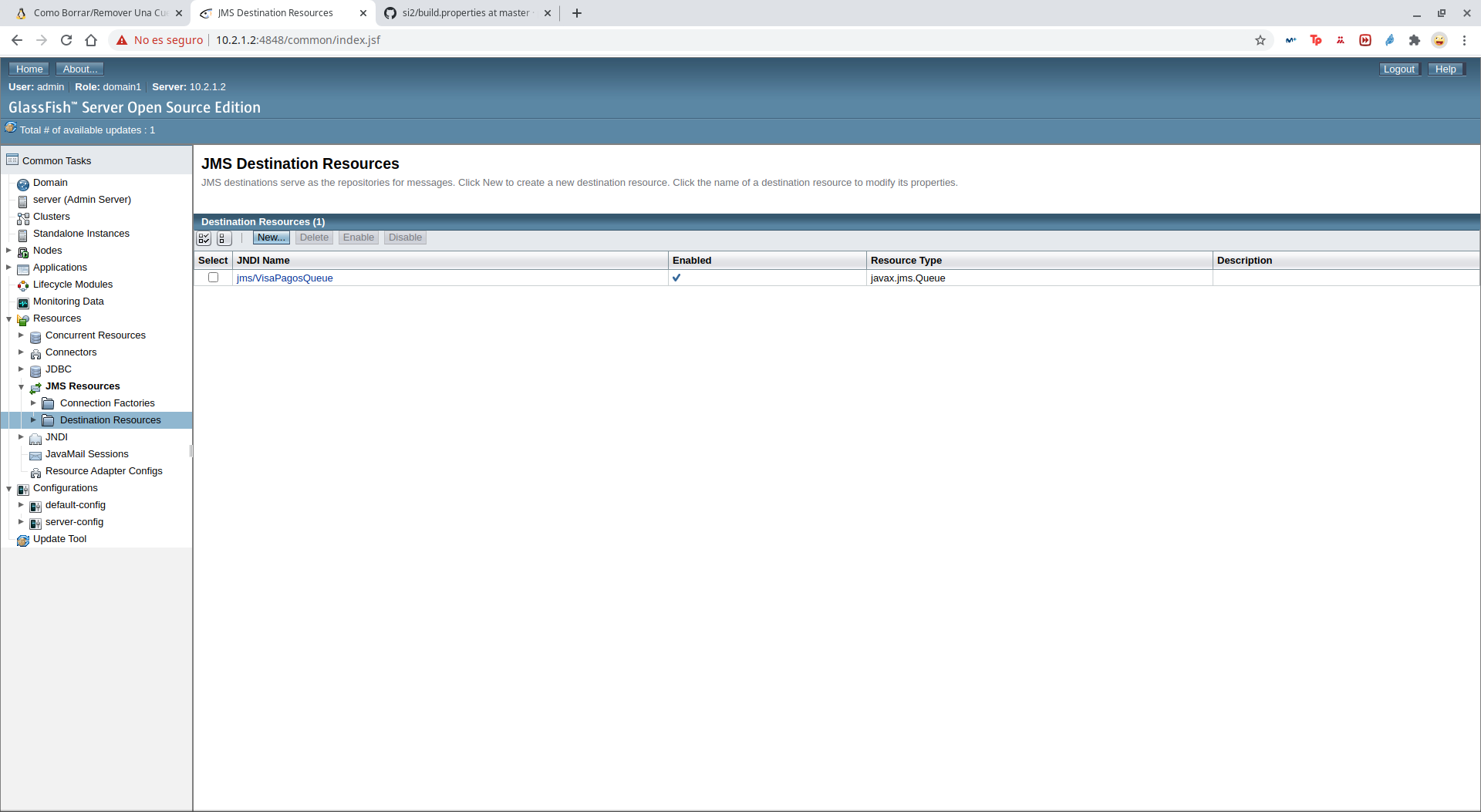
***Ejercicio 13:***

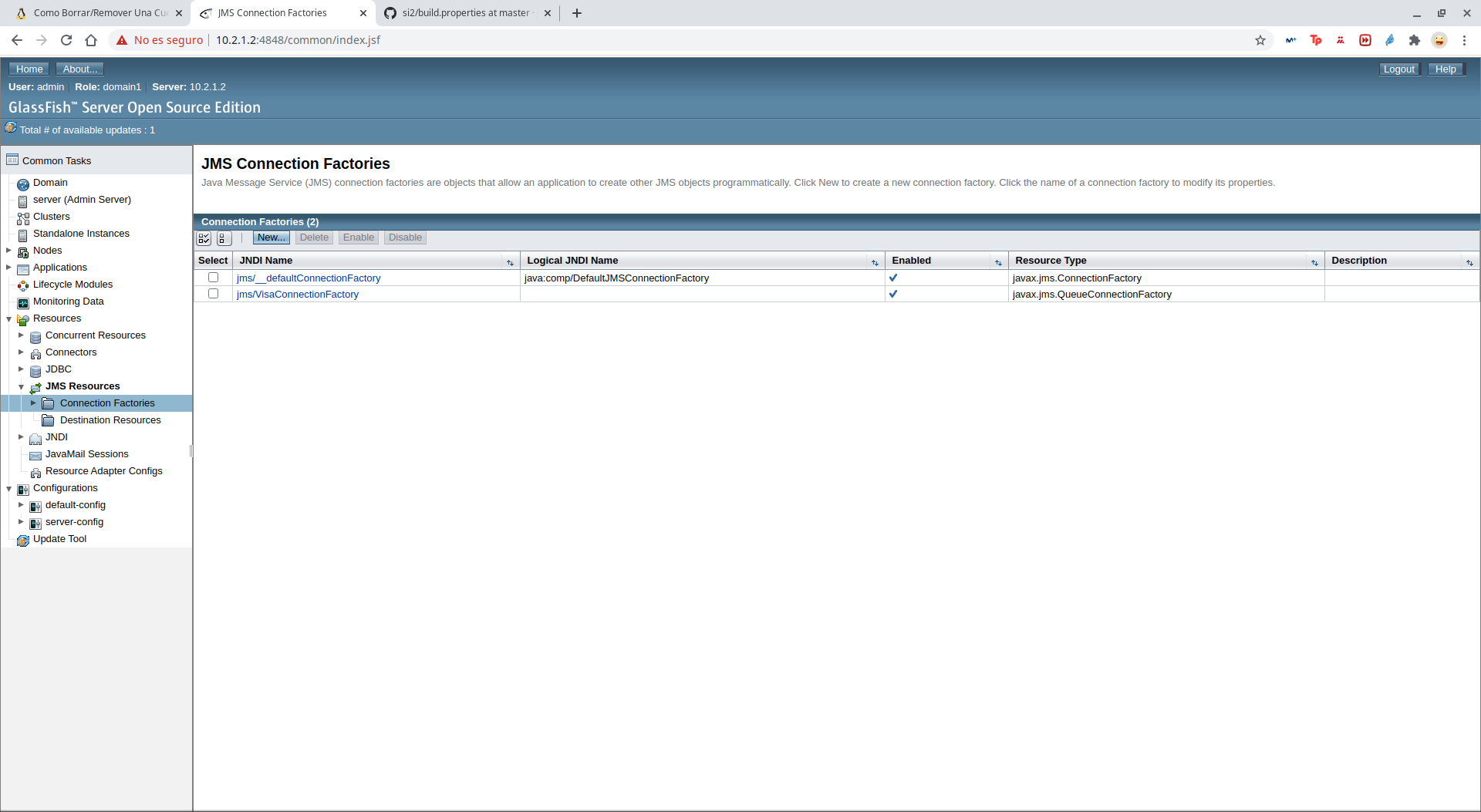
**Automatice la creación de los recursos JMS (cola y factoría de conexiones) en el build.xml y jms.xml. Para ello, indique en jms.properties los nombres de ambos y el Physical Destination Name de la cola de acuerdo a los valores asignados en los ejercicios 9 y 10. Recuerde también asignar las direcciones IP adecuadas a las variables as.host.mdb (build.properties) y as.host.server (jms.properties). ¿Por qué ha añadido esas IPs?**

**Borre desde la consola de administración de Glassfish la connectionFactory y la cola creadas manualmente y ejecute: cd P1-jms ant todo Compruebe en la consola de administración del Glassfish que, efectivamente, los recursos se han creado automáticamente. Incluye una captura de pantalla, donde se muestre la consola de administración con los recursos creados. Revise el fichero jms.xml y anote en la memoria de prácticas cuál es el comando equivalente para crear una cola JMS usando la herramienta asadmin.**

Como nos piden, modificamos los archivos build.properties y jms.properties. Asignamos la IP 10.2.1.2 al as.host.mdb porque es la IP donde hemos ido trabajando y donde está ubicada la aplicación. Además, en el fichero jms.properties le asignamos el VisaPagosQueue al jsm.name, el VisaConnectionFactory a jms.factoryname, Visa a jms.physname y la IP 10.2.1.2 al as.host.server pues es básicamente la IP que usamos para el servidor.

Ejecutamos los comandos del enunciado y vemos que la cola de mensajes se ha creado en el admin:





El comando final sería:

asadmin --user admin --passwordfile passwordfile --host 10.2.1.2 --port 4848 create-jms-resource --restype javax.jms.QueueConnectionFactory --enabled=true --property Name=VisaPagosQueue jms/VisaPagosQueue

***Ejercicio 14:***

**Importante: Detenga la ejecución del MDB con la consola de administración para poder realizar satisfactoriamente el siguiente ejercicio (check de ‘Enabled’ en Applications/P1-jmsmdb y guardar los cambios).**