		Escuela Politécnica Superior Ingeniería Informática Prácticas de Sistemas Informáticos 2			
Grupo	1401	Práctica	1a	Fecha	24/02/2020
Alumno/a		Pelayo, Alvaredo, Victoria			
Alumno/a		Benjumeda, Rubio, Jose			

Ejercicio 1

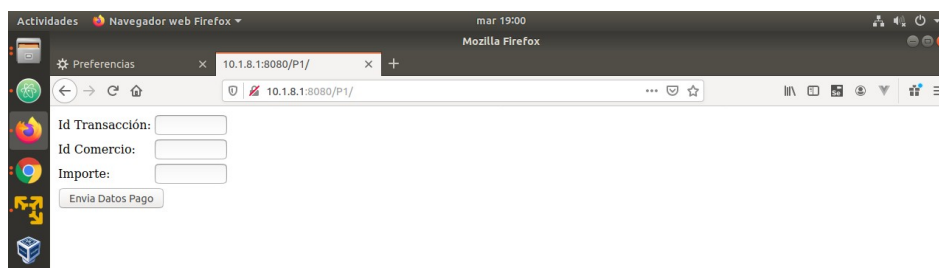
Para realizar este ejercicio, con VMWARE creamos una máquina virtual utilizando el fichero descargado de moodle. Con el botón “Edit virtual machine settings” asignamos 1 GB de RAM y 2 procesadores. Luego hacemos clic en “Power On”.

Ahora, como se indica en el enunciado, justo arriba del ejercicio 1, hay que cambiar ciertos parámetros: el primero es la IP de servidor de aplicaciones, que encontramos en el fichero build.properties en la variable as.host. Le asignamos el valor de nuestra ip: 10.1.8.1. El segundo es la dirección del servidor postgresql que maneja la base de datos. Como se pide que esté en la misma máquina, le damos la misma ip. Este parámetro está en el fichero postgresql.properties, en la variable db.host. Por último, hay que configurar la dirección del servidor de aplicaciones que utiliza el DataSource y el pool de conexiones para acceder a la base de datos, que es la manera que tiene nuestra aplicación de interactuar con la base de datos de manera simplificada. Le damos la dirección de donde esté la aplicación que lo va a utilizar, es decir, otra vez 10.1.8.1. Este parámetro está en el mismo fichero postgresql.properties en la variable db.client.host.

A continuación, abrimos una terminal en la máquina host, y ejecutamos **ssh si2@10.1.8.1** e introducimos la contraseña 2020sid0s, para controlar la máquina virtual desde ahí. En esa misma terminal, que a partir de ahora llamaremos terminal del sistema invitado, ejecutamos el servidor Glassfish con **asadmin start-domain domain1**.

Desde otra terminal, navegamos hasta el directorio de P1-base, donde tenemos el fichero build.xml, y ahí ejecutamos **ant todo**. Con esto estamos creando la aplicación P1.

Desde Mozilla Firefox, vamos a la url <http://10.1.8.1:8080/P1/>, y accedemos al siguiente formulario:



Lo rellenamos con 1, 1, y 15 (números aleatorios). Accedemos al siguiente formulario:

Victoria Pelayo Alvaredo
José Benjumedá Rubio

Preferencias Sistema de Pago con tarjeta +

10.1.8.1:8080/P1/comienzapago

Pago con tarjeta

Numero de visa:

Titular:

Fecha Emisión:

Fecha Caducidad:

CVV2:

Pagar

Id Transacción: 1
Id Comercion: 1
Importe: 15.0

Prácticas de Sistemas Informáticos II

Para ver datos de alguna tarjeta, abrimos Tora. En el cuadro de diálogo que aparece nada más abrirlo, seleccionamos:

Connection provider “Postgresql”

Username: alumnodb

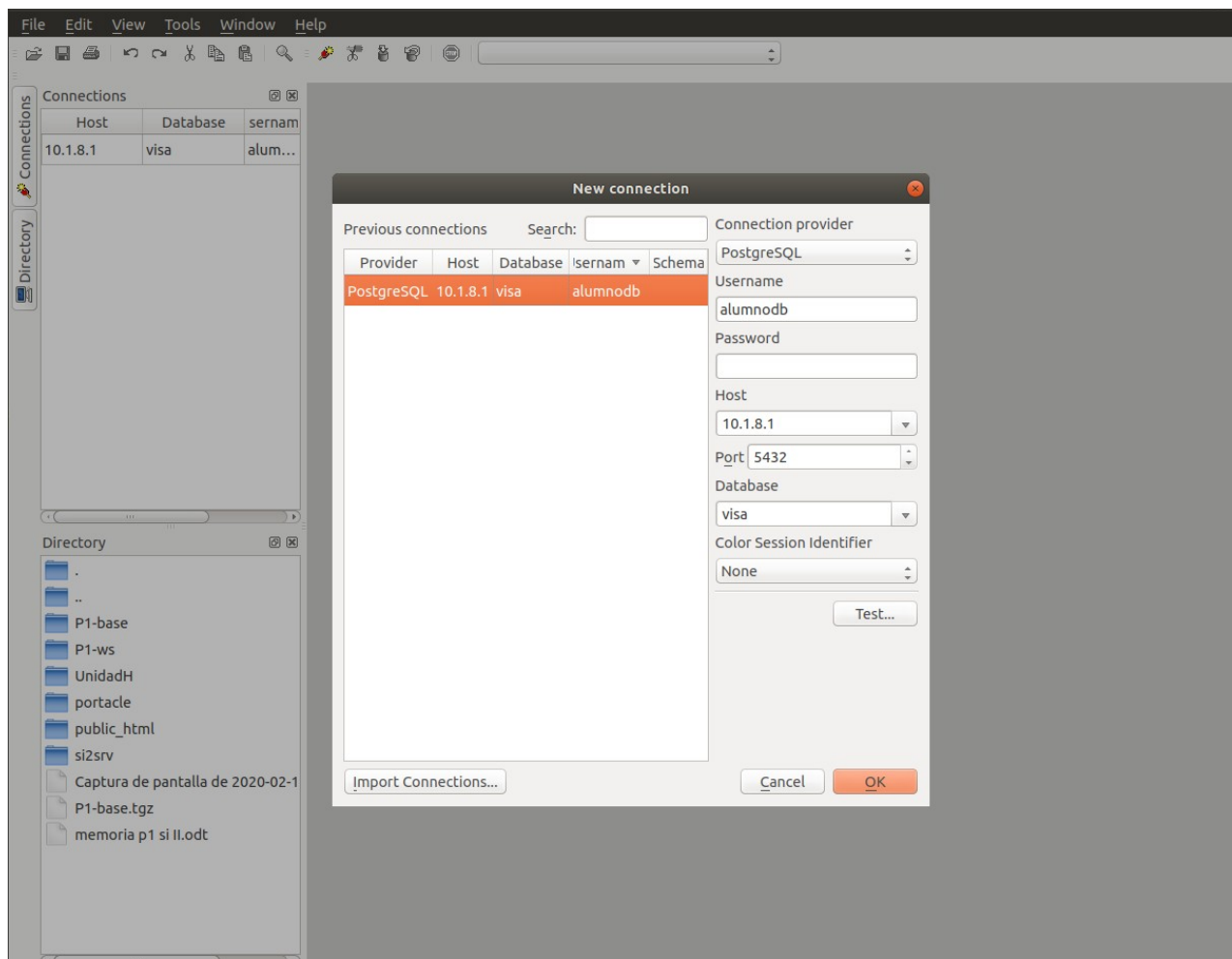
Password:

Host: 10.1.8.1

Port 5432

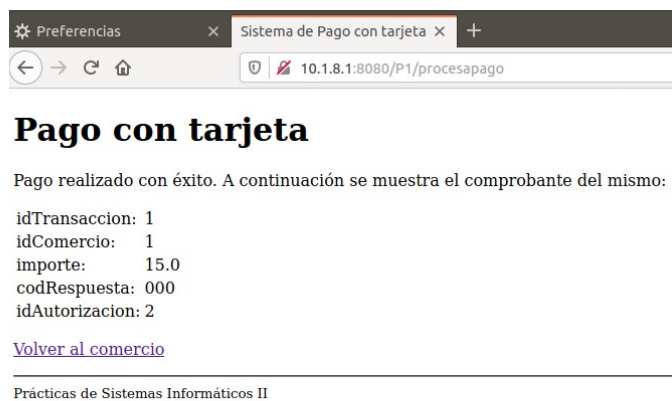
Database: visa

Además, la seleccionamos en el cuadro de la izquierda:

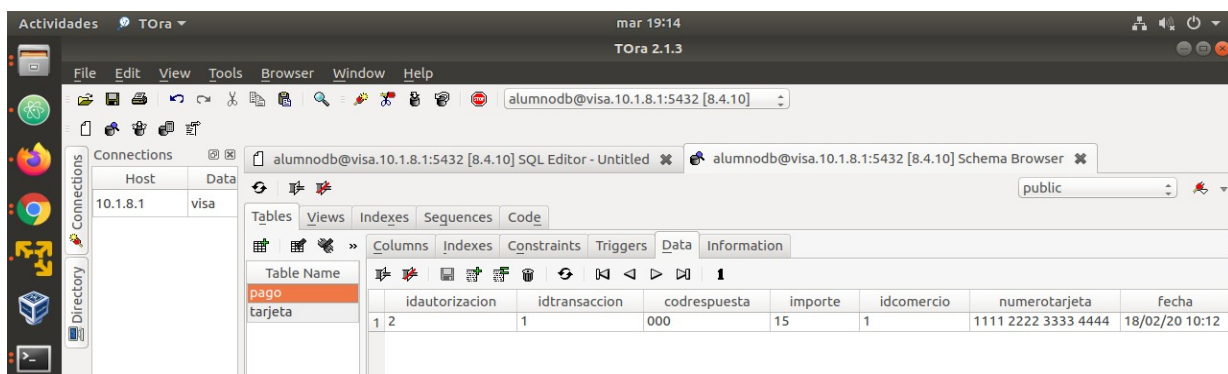


Tras presionar “OK”, en el menú superior seleccionamos Tools → Schema Browser, y vemos que nos aparecen las dos tablas existentes: pago y tarjeta. En la pestaña data podemos ver los datos contenidos en cada tabla: pago está vacía y tarjeta contiene muchas tarjetas. Utilizamos los datos de la primera que aparece, la de Jose Garcia, para rellenar el formulario. Luego clicamos en “Pagar” y

Victoria Pelayo Alvaredo
José Benjumeda Rubio
obtenemos la pantalla de confirmación de pago:



En tora, si seleccionamos la tabla pago y la pestaña data, vemos que se ha creado una fila con los datos de nuestro pago:



Finalmente, borramos el pago. Accedemos a la url <http://10.1.8.1://8080/P1/testbd.jsp>, y en el formulario de borrado de pagos, introducimos el id de comercio 1 y pulsamos borrar: build.



Por último, en tora, en la tabla pago y en la pestaña Data, podemos ver que el pago ha desaparecido.

Ejercicio 2

Para este ejercicio hemos tenido que modificar tres variables del fichero DBTester.java. Los cambios realizados son los siguientes:

- JDBC_DRIVER = "org.postgresql.Driver"
- JDBC_CONNECTIONSTRING = "jdbc:postgresql://10.1.8.1:5432/visa"
- JDBC_USER = "alumnodb"
- JDBC_PASSWORD = ""

Hemos tenido que cambiar el nombre del driver, el *JDBC connection string*, y el nombre de usuario y contraseña.

Para realizar este ejercicio hemos consultado el apéndice 10 dónde se explican detalles para obtener una conexión correcta.

A continuación añadimos unas capturas para verificar el funcionamiento:

Primero probamos a introducir una tarjeta y comprobamos que el pago se ha realizado con éxito:

The screenshot shows a web browser window with the address bar displaying "10.1.8.1:8080/P1/testbd.jsp". The page title is "Pago con tarjeta". The main content area is divided into three sections: "Proceso de un pago", "Consulta de pagos", and "Borrado de pagos".

Proceso de un pago

Id Transacción:
Id Comercio:
Importe:
Número de visa:
Titular:
Fecha Emisión:
Fecha Caducidad:
CVV2:
Modo debug: ☐ True ☒ False
Direct Connection: ☒ True ☐ False
Use Prepared: ☐ True ☒ False

Consulta de pagos

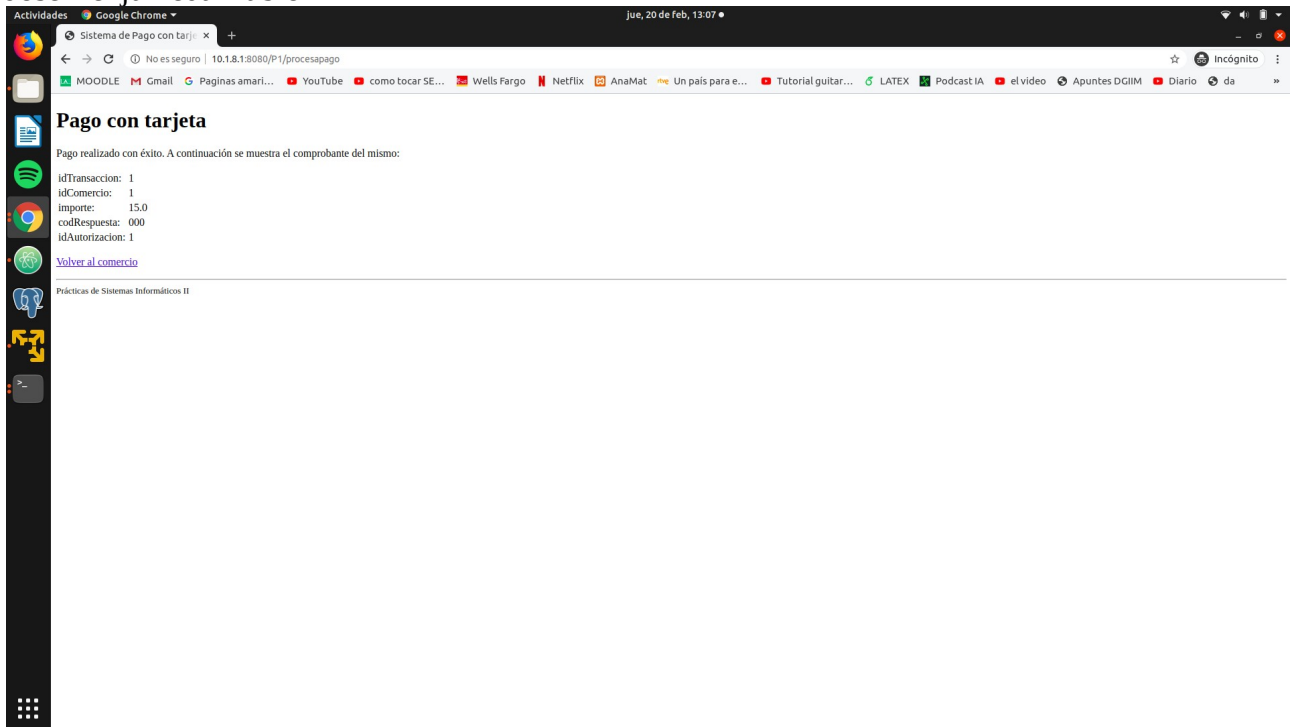
Id Comercio:

Borrado de pagos

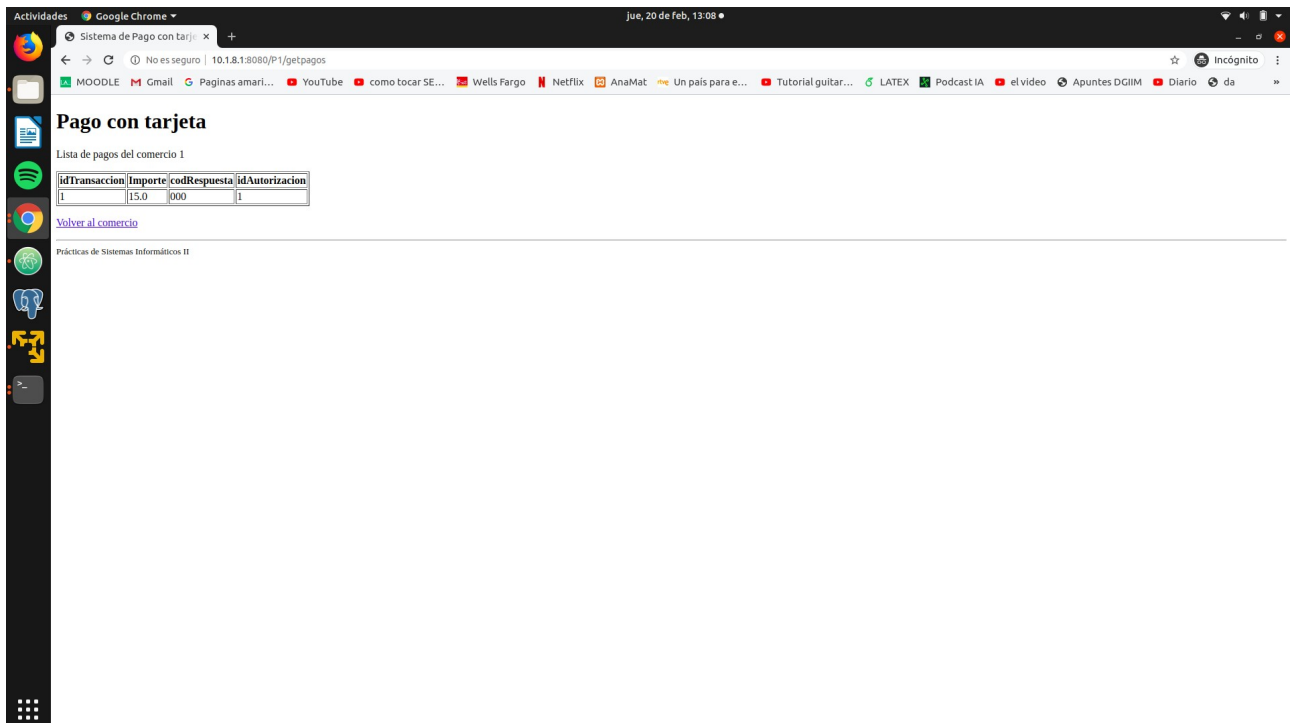
Id Comercio:

Prácticas de Sistemas Informáticos II

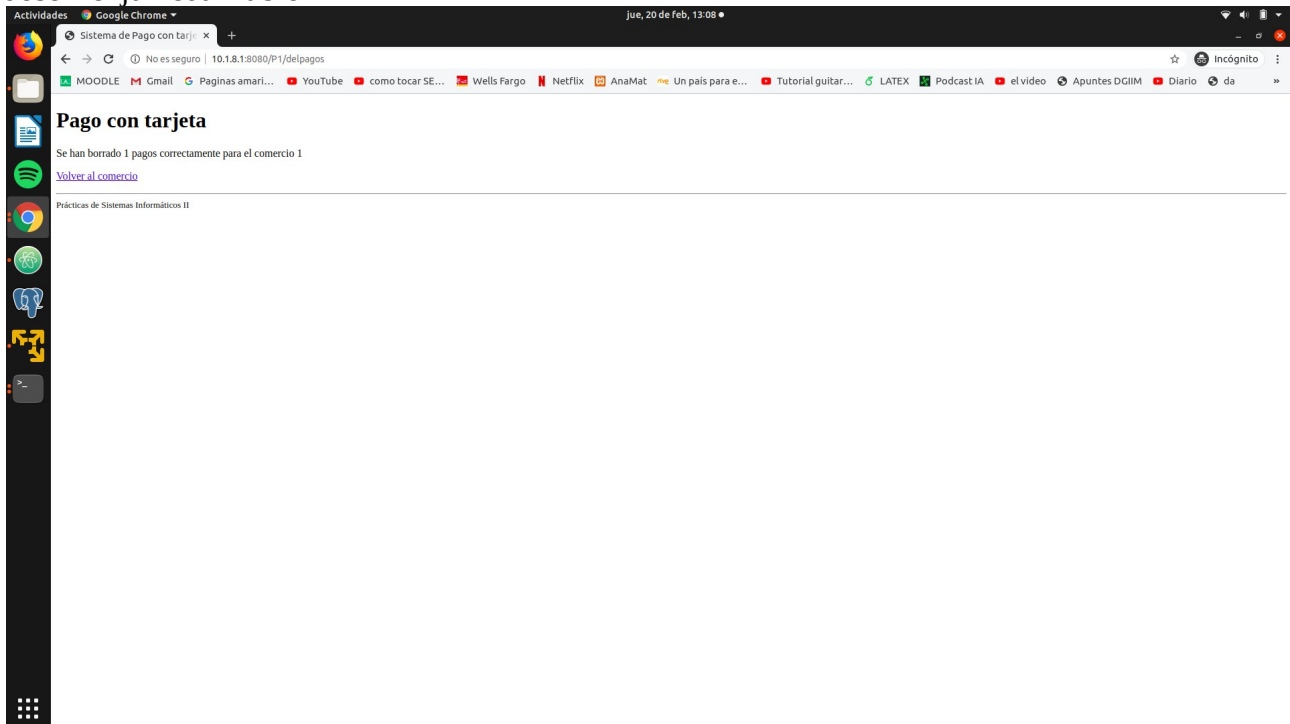
Victoria Pelayo Alvaredo
José Benjumedá Rubio



Posteriormente probamos a listar los pagos con ese idComercio(solo aparecerá el que acabamos de realizar) y luego probaremos a borrarlo. Comprobamos que ambas operaciones se realizan con éxito:



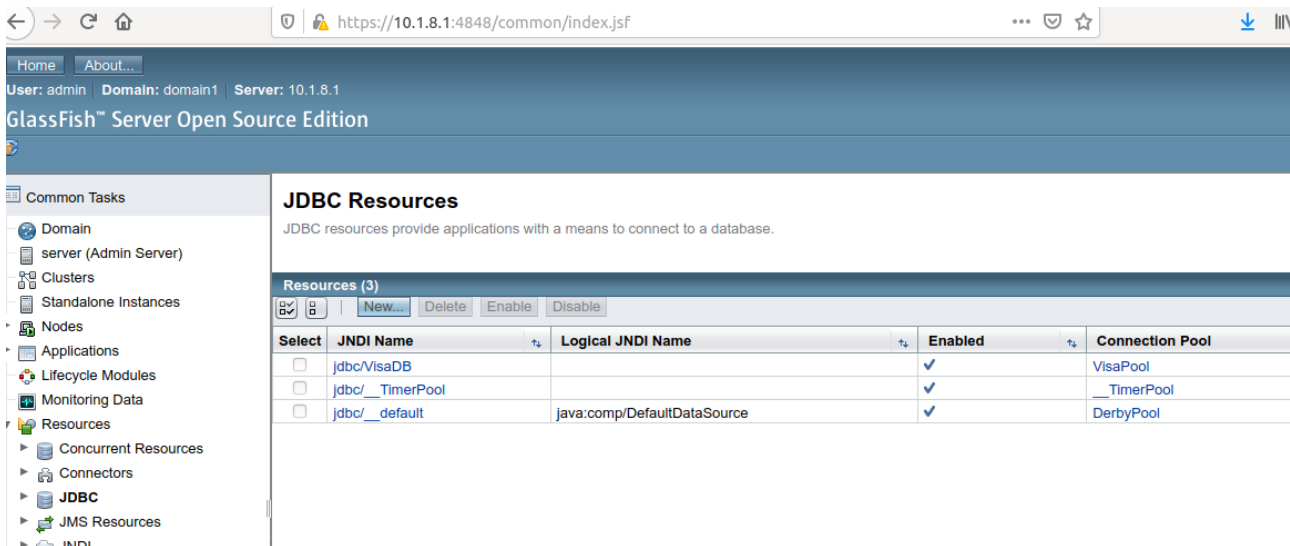
Victoria Pelayo Alvaredo
José Benjumedá Rubio



Ejercicio 3

Primero hemos consultado el fichero *postgresql.properties* y vemos que el nombre del recurso JDBC correspondiente al *DataSource* es “jdbc/VisaDB” y el nombre del pool es “VisaPool”.

Primero comprobamos los recursos JDBC:



Comprobamos el pool de conexiones:

Victoria Pelayo Alvaredo
José Benjumedá Rubio

The screenshot shows the GlassFish Server Open Source Edition web console. The browser address bar displays `https://10.1.8.1:4848/common/index.jsf`. The console header shows the user is 'admin' on 'domain1' with server version '10.1.8.1'. The left sidebar contains a tree view of configuration categories: Common Tasks, Domain (server, clusters, standalone instances, nodes), Applications, Lifecycle Modules, Monitoring Data, Resources (Concurrent Resources, Connectors, JDBC, JMS Resources, JNDI, JavaMail Sessions, Resource Adapter Configs), and Configurations. The main content area is titled 'JDBC Connection Pools' and includes a description: 'To store, organize, and retrieve data, most applications use relational databases. Java EE applications access relational databases through the JDBC API. Before an application can access database, it must get a connection.' Below this is a table of existing pools:

Select	Pool Name	Resource Type	Classname	Description
<input type="checkbox"/>	DerbyPool	javax.sql.DataSource	org.apache.derby.jdbc.ClientDataSource	
<input type="checkbox"/>	VisaPool	javax.sql.ConnectionPoolDataSource	org.postgresql.ds.PGConnectionPoolDataSource	
<input type="checkbox"/>	TimerPool	javax.sql.XADataSource	org.apache.derby.jdbc.EmbeddedXADataSource	

Después realizamos un ping JDBC a la base de datos

The screenshot shows the 'Edit JDBC Connection Pool' configuration page for the 'VisaPool'. The page has tabs for 'General', 'Advanced', and 'Additional Properties', with 'General' selected. A yellow banner at the top indicates 'Ping Succeeded'. Below the title, there is a description of a JDBC connection pool and buttons for 'Load Defaults', 'Flush', and 'Ping'. A legend indicates that an asterisk (*) denotes a required field.

General Settings

Pool Name: VisaPool

Resource Type: `javax.sql.ConnectionPoolDataSource`
Must be specified if the datasource class implements more than 1 of the interface.

Datasource Classname: `org.postgresql.ds.PGConnectionPoolDataSource`
Vendor-specific classname that implements the DataSource and/or XADataSource APIs

Driver Classname:
Vendor-specific classname that implements the java.sql.Driver interface.

Ping: ☐ **Enabled**
When enabled, the pool is pinged during creation or reconfiguration to identify and warn of any erroneous values for its attributes

Deployment Order:
Specifies the loading order of the resource at server startup. Lower numbers are loaded first.

Description:

Pool Settings

Initial and Minimum Pool Size: Connections
Minimum and initial number of connections maintained in the pool

Maximum Pool Size: Connections
Maximum number of connections that can be created to satisfy client requests

Y consultamos los valores indicados en el enunciado (***Initial and Minimum Pool Size, Maximum Pool Size, Pool Resize Quantity, Idle Timeout, Max Wait Time***).

Pool Settings

Initial and Minimum Pool Size:	<input type="text" value="8"/>	Connections
Minimum and initial number of connections maintained in the pool		
Maximum Pool Size:	<input type="text" value="32"/>	Connections
Maximum number of connections that can be created to satisfy client requests		
Pool Resize Quantity:	<input type="text" value="2"/>	Connections
Number of connections to be removed when pool idle timeout expires		
Idle Timeout:	<input type="text" value="300"/>	Seconds
Maximum time that connection can remain idle in the pool		
Max Wait Time:	<input type="text" value="60000"/>	Milliseconds
Amount of time caller waits before connection timeout is sent		

Crear una conexión a la base de datos es una operación costosa, por tanto tener un número inicial muy bajo si se van a tener muchas conexiones sería muy ineficiente. Tener un número mínimo muy alto, si se van a tener pocas conexiones, también lo sería al principio.

El número máximo si es muy bajo crearía una cola de espera en el servidor que haría que bajase el rendimiento. Sin embargo, si es demasiado alto sobrecargaría al servidor de manera que no podría atender a las conexiones de manera aceptable, lo que haría que también bajase el rendimiento.

Si el timeout fuese muy bajo haría que estuviésemos conectando y desconectando constantemente.

Si fuese demasiado alto podría producirse que alguien monopolizase la conexión.

El *max wait* si fuese muy bajo podría cortar conexiones más lentas.

Ejercicio 4

La función *compruebaTarjeta* vemos que llama a otra función llamada *getQryCompruebaTarjeta* que es la que se encarga de realizar la consulta a la base de datos. Esta consulta es la siguiente:

```
"select * from tarjeta "  
    + "where numeroTarjeta='" + tarjeta.getNumero()  
    + "' and titular='" + tarjeta.getTitular()  
    + "' and validaDesde='" + tarjeta.getFechaEmision()  
    + "' and validaHasta='" + tarjeta.getFechaCaducidad()  
    + "' and codigoVerificacion='" +  
tarjeta.getCodigoVerificacion() + "'";
```

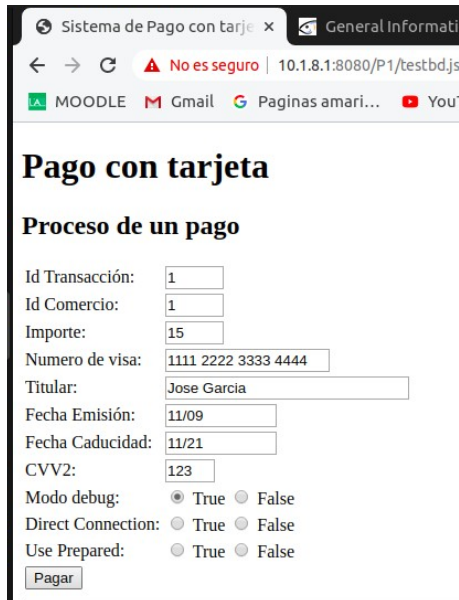
La función *realizaPago* vemos que llama a la función *getQryInsertPago* que es la encargada de realizar la consulta para insertar un pago en la base de datos. La consulta que se realiza es la siguiente:

```
"insert into pago("  
    + "idTransaccion,"  
    + "importe,idComercio,"  
    + "numeroTarjeta)"  
    + " values ("  
    + "'" + pago.getIdTransaccion() + "',"  
    + pago.getImporte() + ","  
    + "'" + pago.getIdComercio() + "',"  
    + "'" + pago.getTarjeta().getNumero() + "'"  
    + ")";
```


Victoria Pelayo Alvaredo
José Benjumedá Rubio
Ejercicio 5

Analizamos el código y vemos que se utiliza en los catch para informar de las excepciones y también se utiliza para informar de las queries que se van a hacer, nos informa de por “dónde” se está ejecutando la aplicación. Hay 12 llamadas a errorLog más la definición.

Ejecutamos un pago con la opción debug activada:



Pago con tarjeta

Proceso de un pago

Id Transacción:

Id Comercio:

Importe:

Numero de visa:

Titular:

Fecha Emisión:

Fecha Caducidad:

CVV2:

Modo debug: ☒ True ☐ False

Direct Connection: ☐ True ☐ False

Use Prepared: ☐ True ☐ False



Pago con tarjeta

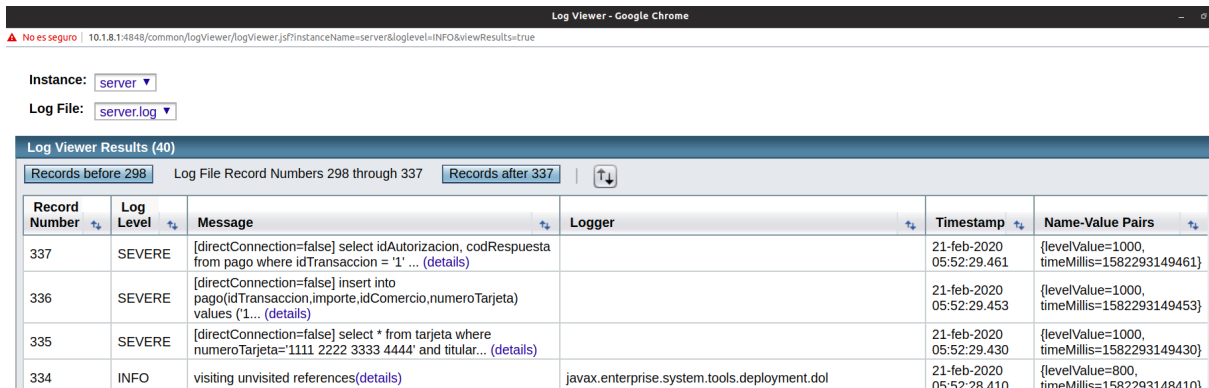
Pago realizado con éxito. A continuación se muestra el comprobante del mismo:

idTransaccion: 1
idComercio: 1
importe: 15.0
codRespuesta: 000
idAutorizacion: 2

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

Comprobamos que en el fichero log aparecen nuevos registros. En concreto estos 4 son los debidos al proceso de este pago:



Record Number	Log Level	Message	Logger	Timestamp	Name-Value Pairs
337	SEVERE	[directConnection=false] select idAutorizacion, codRespuesta from pago where idTransaccion = '1' ... (details)		21-feb-2020 05:52:29.461	{levelValue=1000, timeMillis=1582293149461}
336	SEVERE	[directConnection=false] insert into pago(idTransaccion,importe,idComercio,numeroTarjeta) values ('1...' (details)		21-feb-2020 05:52:29.453	{levelValue=1000, timeMillis=1582293149453}
335	SEVERE	[directConnection=false] select * from tarjeta where numeroTarjeta='1111 2222 3333 4444' and titular... (details)		21-feb-2020 05:52:29.430	{levelValue=1000, timeMillis=1582293149430}
334	INFO	visiting unvisited references(details)	javax.enterprise.system.tools.deployment.dol	21-feb-2020 05:52:28.410	{levelValue=800, timeMillis=1582293148410}

Ejercicio 6

En este ejercicio nos piden que modifiquemos el fichero visaDAOWS.java para que los métodos listados sean publicados como métodos de servicio.

Primero hemos añadido “@WebService()” en la clase VisaDAOWS y DbTester para indicar que implementan un servicio Web.

```
@WebService()  
public class VisaDAOWS extends DBTester {
```

Victoria Pelayo Alvaredo
José Benjumedá Rubio
@WebService()
public class DBTester {

Ejemplo de la modificación que hemos tenido que realizar en `compruebaTarjeta`:

- Hemos añadido `@WebMethod` y `@WebParam`.

```
@WebMethod(operationName = "compruebaTarjeta")  
public boolean compruebaTarjeta(@WebParam(name = "tarjeta") TarjetaBean tarjeta) {
```

Esta modificación se ha realizado de manera análoga en el resto de métodos.

```
@WebMethod(operationName = "realizaPago")  
public synchronized PagoBean realizaPago(@WebParam(name = "pago") PagoBean pago)  
{
```

```
@WebMethod(operationName = "isPrepared")  
public boolean isPrepared() {
```

```
@WebMethod(operationName = "setPrepared")  
public void setPrepared(@WebParam(name = "prepared") boolean prepared) {
```

```
@WebMethod(operationName = "isDebug")  
public boolean isDebug() {
```

```
@WebMethod(operationName = "setDebug")  
public void setDebug(@WebParam(name = "debug") boolean debug) {
```

```
@WebMethod(exclude=true)  
public void setDebug(String debug) {
```

`isDirectConnection` y `setDirectConnection` los hemos implementado en `VisaDAOWS`, la clase hija.

```
@WebMethod(operationName = "isDirectConnection")  
@Override  
public boolean isDirectConnection() {
```

```
@WebMethod(operationName = "setDirectConnection")  
@Override  
public void setDirectConnection(@WebParam(name="directConnection") boolean  
directConnection) {
```

Las modificaciones de estos métodos en la clase padre(*DBTester*) son:

```
@WebMethod(operationName = "isDirectConnection")  
public boolean isDirectConnection() {
```

```
@WebMethod(operationName = "setDirectConnection")  
public void setDirectConnection(@WebParam(name="directConnection") boolean  
directConnection) {
```

Por último destacar que en la función *realizaPago()* se ha modificado el parámetro de retorno para que devuelva un pago, en lugar de simplemente un booleano en función de si ha habido errores o no.

Esto se debe a que nuestro objetivo es, además de publicar las funciones de visadao como servicios web, tener una modularización mayor, para que puedan utilizarse nuestras funciones individualmente en otro contexto, es decir, que ahora puede que en otro programa completamente ajeno al nuestro, se quiera utilizar nuestro método *realizaPago()*, y en ese otro programa es probable que se necesiten los datos del pago no solo para guardarlos en la base de datos en que se guardan dentro de nuestra función, sino para enviarlos o guardarlos en otro sitio.

Por esto es por lo que al modularizar y publicar *realizarPago()* como servicio independiente, nos interesa más que devuelva el pago, y no solo un booleano que únicamente indique si se ha guardado en la base de datos o no.

Ejercicio 7

Para este ejercicio, se ha realizado la copia de ficheros y directorios de la carpeta P1-base a P1-ws siguiendo la distribución que se indica en el enunciado de la práctica. Seguimos en el proceso de distribuir la tarea de realizar un pago. Nos interesa aislar las funciones que acceden a datos, de manera que donde antes había simplemente una parte más de nuestro servidor, ahora vamos a tener otro cliente y otro servidor, porque, dentro de nuestro servidor, vamos a distribuir una tarea, y esto consiste en que una parte la realice, el servidor, y otra parte le pida que la realice, el cliente.

La parte cliente (cliente VisaDAO) serán los servlets de nuestra aplicación que, de aquí en adelante, van a estar desacoplados del acceso a los datos. Este cliente utilizará el nuevo y más pequeño servicio web para terminar de realizar cada pago.

La parte servidor (servidor VisaDAO) serán las funciones que realizan acceso a datos, es decir, tanto los métodos para consultar si una tarjeta es válida como el que registra el pago en la base de datos.

Ahora, nuestro servidor (el principal, no este nuevo y pequeño) puede repartirse en distintas máquinas, y de hecho lo vamos a hacer. En una desplegaremos la base de datos y el servidor VisaDAO, y en otra el cliente VisaDAO. Tendrán IPs 10.1.8.1 y 10.1.8.2 respectivamente. Para indicar esto, editamos el fichero *build.properties*, modificando las líneas:

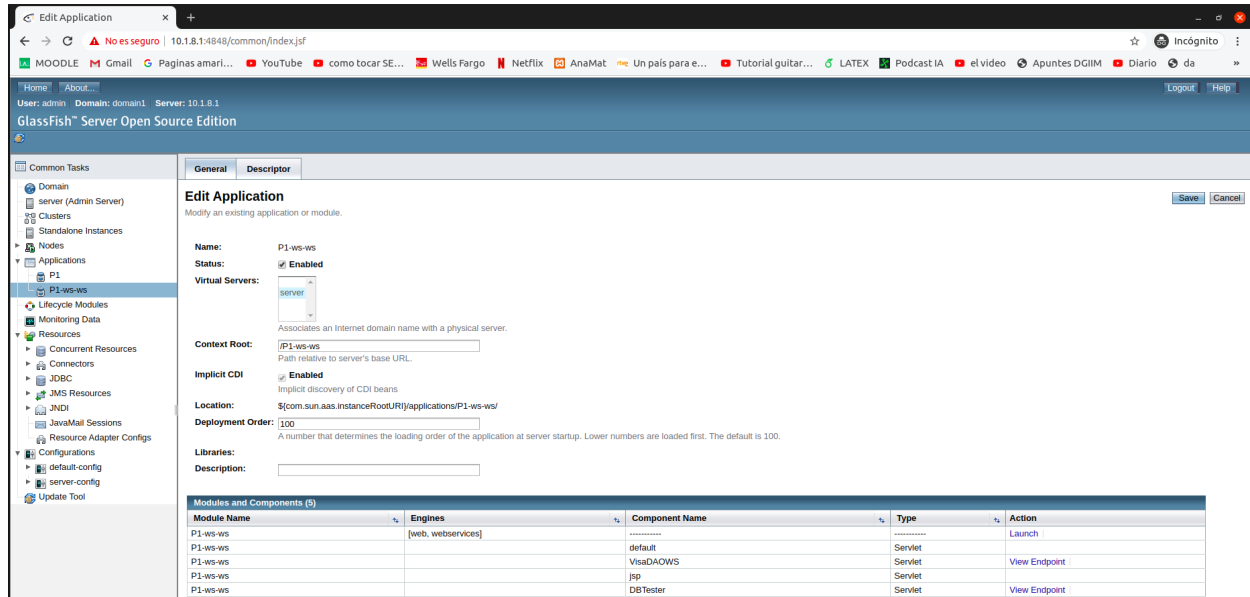
```
as.host.client=10.1.8.2  
as.host.server=10.1.8.1
```

Lanzamos la máquina virtual con ip 10.1.8.1 y que contiene el servidor VisaDAO (de ahora en adelante nos referiremos a ella como VMS). Desde otra terminal ejecutamos el primer comando, que nos compilará el servidor: **ant compilar-servicio**.

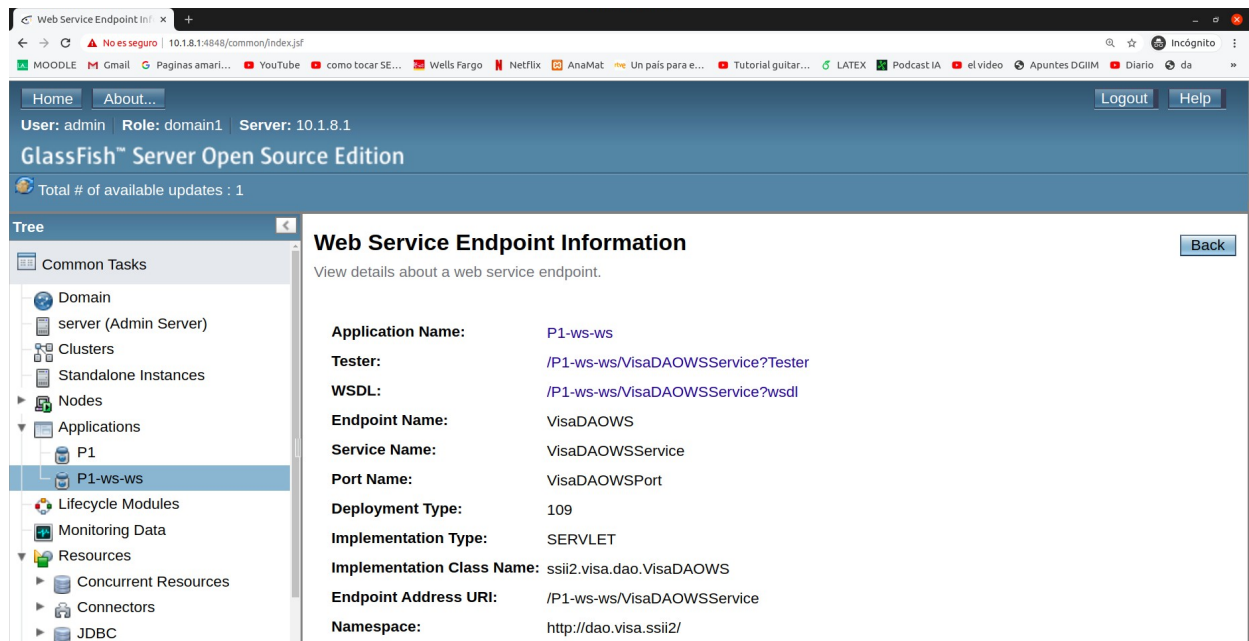
Victoria Pelayo Alvaredo
José Benjumedá Rubio

Lo siguiente que tenemos que hacer es empaquetar los archivos del servicio en un archivo .war, para lo que utilizaremos **ant empaquetar-servicio**. Comprobamos que en dist/server se ha creado P1-ws-ws.war.

Finalmente, desplegamos el servicio con **ant desplegar-servicio**, y comprobamos, en la url <http://10.1.8.1:4848> que se ha desplegado correctamente:



Y haciendo clic en “View Endpoint” de la fila VisaDOWS, vemos la siguiente página:



Accedemos al WSDL con la url <http://10.1.8.1:8080/P1-ws-ws/VisaDAOWSService?wsdl>.

Código del WSDL

```
<!--
Published by JAX-WS RI (http://jax-ws.java.net). RI's version is Metro/2.3.2-b608
(trunk-7979; 2015-01-21T12:50:19+0000) JAXWS-RI/2.2.11-b150120.1832
JAXWS-API/2.2.12 JAXB-RI/2.2.12-b141219.1637 JAXB-API/2.2.13-b141020.1521 svn-
revision#unknown.
-->
<!--
Generated by JAX-WS RI (http://jax-ws.java.net). RI's version is Metro/2.3.2-b608
(trunk-7979; 2015-01-21T12:50:19+0000) JAXWS-RI/2.2.11-b150120.1832
JAXWS-API/2.2.12 JAXB-RI/2.2.12-b141219.1637 JAXB-API/2.2.13-b141020.1521 svn-
revision#unknown.
-->
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd" xmlns:wsp="http://www.w3.org/ns/ws-policy" xmlns:wsp1_
2="http://schemas.xmlsoap.org/ws/2004/09/policy" xmlns:wsam="http://www.w3.org/
2007/05/addressing/metadata" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xm
lns:tns="http://dao.visa.ssii2/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmln
s="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://dao.visa.ssii2/" name
="VisaDAOWSService">
<types>
<xsd:schema>
<xsd:import namespace="http://dao.visa.ssii2/" schemaLocation="http://
10.1.8.1:8080/P1-ws-ws/VisaDAOWSService?xsd=1"/>
</xsd:schema>
</types>
<message name="getPagos">
<part name="parameters" element="tns:getPagos"/>
</message>
<message name="getPagosResponse">
<part name="parameters" element="tns:getPagosResponse"/>
</message>
<message name="delPagos">
<part name="parameters" element="tns:delPagos"/>
</message>
<message name="delPagosResponse">
<part name="parameters" element="tns:delPagosResponse"/>
</message>
<message name="isPrepared">
<part name="parameters" element="tns:isPrepared"/>
</message>
<message name="isPreparedResponse">
<part name="parameters" element="tns:isPreparedResponse"/>
</message>
<message name="setPrepared">
<part name="parameters" element="tns:setPrepared"/>
</message>
<message name="setPreparedResponse">
<part name="parameters" element="tns:setPreparedResponse"/>
</message>
<message name="errorLog">
<part name="parameters" element="tns:errorLog"/>
</message>
<message name="errorLogResponse">
<part name="parameters" element="tns:errorLogResponse"/>
</message>
<message name="setDirectConnection">
<part name="parameters" element="tns:setDirectConnection"/>
</message>
<message name="setDirectConnectionResponse">
<part name="parameters" element="tns:setDirectConnectionResponse"/>
```

Victoria Pelayo Alvaredo

José Benjumeda Rubio

```
</message>
<message name="compruebaTarjeta">
<part name="parameters" element="tns:compruebaTarjeta"/>
</message>
<message name="compruebaTarjetaResponse">
<part name="parameters" element="tns:compruebaTarjetaResponse"/>
</message>
<message name="realizaPago">
<part name="parameters" element="tns:realizaPago"/>
</message>
<message name="realizaPagoResponse">
<part name="parameters" element="tns:realizaPagoResponse"/>
</message>
<message name="isDirectConnection">
<part name="parameters" element="tns:isDirectConnection"/>
</message>
<message name="isDirectConnectionResponse">
<part name="parameters" element="tns:isDirectConnectionResponse"/>
</message>
<message name="isDebug">
<part name="parameters" element="tns:isDebug"/>
</message>
<message name="isDebugResponse">
<part name="parameters" element="tns:isDebugResponse"/>
</message>
<message name="setDebug">
<part name="parameters" element="tns:setDebug"/>
</message>
<message name="setDebugResponse">
<part name="parameters" element="tns:setDebugResponse"/>
</message>
<message name="getDirectConnectionCount">
<part name="parameters" element="tns:getDirectConnectionCount"/>
</message>
<message name="getDirectConnectionCountResponse">
<part name="parameters" element="tns:getDirectConnectionCountResponse"/>
</message>
<message name="getDSNConnectionCount">
<part name="parameters" element="tns:getDSNConnectionCount"/>
</message>
<message name="getDSNConnectionCountResponse">
<part name="parameters" element="tns:getDSNConnectionCountResponse"/>
</message>
<portType name="VisaDAOWS">
<operation name="getPagos">
<input wsam:Action="http://dao.visa.ssii2/VisaDAOWS/getPagosRequest" message="tns:
getPagos"/>
<output wsam:Action="http://dao.visa.ssii2/VisaDAOWS/getPagosResponse" message="tn
s:getPagosResponse"/>
</operation>
<operation name="delPagos">
<input wsam:Action="http://dao.visa.ssii2/VisaDAOWS/delPagosRequest" message="tns:
delPagos"/>
<output wsam:Action="http://dao.visa.ssii2/VisaDAOWS/delPagosResponse" message="tn
s:delPagosResponse"/>
</operation>
<operation name="isPrepared">
<input wsam:Action="http://dao.visa.ssii2/VisaDAOWS/isPreparedRequest" message="tn
s:isPrepared"/>
<output wsam:Action="http://dao.visa.ssii2/VisaDAOWS/isPreparedResponse" message="
tns:isPreparedResponse"/>
</operation>
<operation name="setPrepared">
<input wsam:Action="http://dao.visa.ssii2/VisaDAOWS/setPreparedRequest" message="t
ns:setPrepared"/>
```

Victoria Pelayo Alvaredo

José Benjumedá Rubio

```
<output wsam:Action="http://dao.visa.ssii2/VisaDAOWS/setPreparedResponse" message=
"tns:setPreparedResponse"/>
</operation>
<operation name="errorLog">
<input wsam:Action="http://dao.visa.ssii2/VisaDAOWS/errorLogRequest" message="tns:
errorLog"/>
<output wsam:Action="http://dao.visa.ssii2/VisaDAOWS/errorLogResponse" message="tn
s:errorLogResponse"/>
</operation>
<operation name="setDirectConnection">
<input wsam:Action="http://dao.visa.ssii2/VisaDAOWS/setDirectConnectionRequest" me
ssage="tns:setDirectConnection"/>
<output wsam:Action="http://dao.visa.ssii2/VisaDAOWS/setDirectConnectionResponse"
message="tns:setDirectConnectionResponse"/>
</operation>
<operation name="compruebaTarjeta">
<input wsam:Action="http://dao.visa.ssii2/VisaDAOWS/compruebaTarjetaRequest" messa
ge="tns:compruebaTarjeta"/>
<output wsam:Action="http://dao.visa.ssii2/VisaDAOWS/compruebaTarjetaResponse" mes
sage="tns:compruebaTarjetaResponse"/>
</operation>
<operation name="realizaPago">
<input wsam:Action="http://dao.visa.ssii2/VisaDAOWS/realizaPagoRequest" message="t
ns:realizaPago"/>
<output wsam:Action="http://dao.visa.ssii2/VisaDAOWS/realizaPagoResponse" message=
"tns:realizaPagoResponse"/>
</operation>
<operation name="isDirectConnection">
<input wsam:Action="http://dao.visa.ssii2/VisaDAOWS/isDirectConnectionRequest" mes
sage="tns:isDirectConnection"/>
<output wsam:Action="http://dao.visa.ssii2/VisaDAOWS/isDirectConnectionResponse" m
essage="tns:isDirectConnectionResponse"/>
</operation>
<operation name="isDebug">
<input wsam:Action="http://dao.visa.ssii2/VisaDAOWS/isDebugRequest" message="tns:i
sDebug"/>
<output wsam:Action="http://dao.visa.ssii2/VisaDAOWS/isDebugResponse" message="tns
:isDebugResponse"/>
</operation>
<operation name="setDebug">
<input wsam:Action="http://dao.visa.ssii2/VisaDAOWS/setDebugRequest" message="tns:
setDebug"/>
<output wsam:Action="http://dao.visa.ssii2/VisaDAOWS/setDebugResponse" message="tn
s:setDebugResponse"/>
</operation>
<operation name="getDirectConnectionCount">
<input wsam:Action="http://dao.visa.ssii2/VisaDAOWS/
getDirectConnectionCountRequest" message="tns:getDirectConnectionCount"/>
<output wsam:Action="http://dao.visa.ssii2/VisaDAOWS/
getDirectConnectionCountResponse" message="tns:getDirectConnectionCountResponse"/>
</operation>
<operation name="getDSNConnectionCount">
<input wsam:Action="http://dao.visa.ssii2/VisaDAOWS/getDSNConnectionCountRequest"
message="tns:getDSNConnectionCount"/>
<output wsam:Action="http://dao.visa.ssii2/VisaDAOWS/
getDSNConnectionCountResponse" message="tns:getDSNConnectionCountResponse"/>
</operation>
</portType>
<binding name="VisaDAOWSPortBinding" type="tns:VisaDAOWS">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
<operation name="getPagos">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal"/>
```

Victoria Pelayo Alvaredo

José Benjumedá Rubio

```
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
<operation name="delPagos">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
<operation name="isPrepared">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
<operation name="setPrepared">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
<operation name="errorLog">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
<operation name="setDirectConnection">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
<operation name="compruebaTarjeta">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
<operation name="realizaPago">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal"/>
```


Victoria Pelayo Alvaredo

José Benjumedá Rubio

```
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
<operation name="isDirectConnection">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
<operation name="isDebug">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
<operation name="setDebug">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
<operation name="getDirectConnectionCount">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
<operation name="getDSNConnectionCount">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
</binding>
<service name="VisaDAOWSService">
<port name="VisaDAOWSPort" binding="tns:VisaDAOWSPortBinding">
<soap:address location="http://10.1.8.1:8080/P1-ws-ws/VisaDAOWSService"/>
</port>
</service>
</definitions>
```

Fin código del WSDL

WSDL es un protocolo para el intercambio de información entre sistemas distribuidos/descentralizados, basado en xml. Las definiciones contenidas en un fichero WSDL

Victoria Pelayo Alvaredo
José Benjumeda Rubio

describen cómo se accede a un servicio web y a qué operaciones se da soporte.

Un programa cliente que quiera conectarse a un servicio web leerá el documento WSDL para determinar qué funciones están disponibles en el servidor. Además, cualquier tipo de dato no predefinido que utilice el servidor será descrito en este documento, en esquema xml. Entonces, el cliente utilizará el Simple Object Access Protocol para llamar a cualquiera de las funciones listadas.

1 - ¿En qué fichero están definidos los tipos de datos intercambiados con el webservice?

Los tipos de datos intercambiados entre cliente y servidor al utilizar el webservice que acabamos de publicar están descritos en el fichero que encontramos en la url

“<http://10.1.8.1:8080/P1-ws-ws/VisaDAOWSService?xsd=1>”, que encontramos en el fichero wsdl, es un fichero xml.

2 - ¿Qué tipos de datos predefinidos se usan?

Los datos simples/predefinidos que se utilizan son string, int, double y boolean.

3 - ¿Cuáles son los tipos de datos que se definen?

Se definen pagoBean y tarjetaBean.

4 - ¿Qué etiqueta está asociada a los métodos invocados en el webservice?

La etiqueta es operation.

5 - ¿Qué etiqueta describe los mensajes intercambiados en la invocación de los métodos del webservice?

La etiqueta es message.

6 - ¿En qué etiqueta se especifica el protocolo de comunicación con el webservice?

La etiqueta es soap:binding.

7 - ¿En qué etiqueta se especifica la URL a la que se deberá conectar un cliente para acceder al webservice?

La etiqueta soap:address, dentro de la etiqueta Service. Aparece al final del fichero WSDL.

Ejercicio 8

Vamos a modificar el fichero que hacía uso de las funciones de VisaDAO como si fuera una clase local, para que se utilice como servidor remoto. Para esto, tenemos que eliminar la instancia de la clase VisaDAO, pues no tenemos acceso a ella de manera directa, y lo que haremos será utilizar una imagen de este servicio. Es la manera que tenemos, desde el cliente, de hacer referencia a una clase cuya definición no está en él.

El primer paso es añadir estas tres líneas, necesarias en Java donde se quiera usar el servicio remoto:

```
import ssii2.visa.VisaDAOWSService; // Stub generado automáticamente  
import ssii2.visa.VisaDAOWS; // Stub generado automáticamente  
import javax.xml.ws.WebServiceRef;
```

A continuación, cambiamos la declaración directa **VisaDAO dao = new VisaDAO();** por una declaración en dos pasos, que es como debe hacerse la instanciación de la clase remota:

```
VisaDAOWSService service = new VisaDAOWSService();  
VisaDAOWS dao = service. GetVisaDAOWSPort ();
```

Respecto a los cambios debido a que realizaPago ya no devuelva un booleano sino un objeto pago o null.

En caso de que la función no devuelva null, tendremos un nuevo objeto pago, con los datos actualizados, y será este el que guardemos en request. Así es como nos queda la función que guarda el pago:

```
pagoModificado = dao.realizaPago(pago);
    if (pagoModificado == null) {
        enviaError(new Exception("Pago incorrecto"), request, response);
        return;
    }

    request.setAttribute(ComienzaPago.ATTR_PAGO, pagoModificado);
    if (sesion != null) sesion.invalidate();
    reenvia("/pagoexito.jsp", request, response);
    return;
}
```

Como le hemos dado a éste objeto el mismo nombre que al anterior, “dao”, ya no hay que hacer ningún cambio más, pues se utiliza de la misma manera que la instancia de la clase VisaDAO que antes utilizábamos como una clase local. Vemos que el uso del servicio web tiene una gran transparencia, pues cambiando una declaración por otra que incluye tan solo un paso más, podemos seguir trabajando como si el sistema no estuviese distribuido, desde el punto de vista de la complejidad, pero aprovechando las ventajas de que realmente sí que lo está.

Ejercicio 9

Para facilitarnos futuras modificaciones de código en caso de que cambiase la ubicación del servidor, vamos a cambiar nuestro código para que la ip se lea del archivo web.xml, para que, en caso de que cambie, únicamente haya que modificar su valor en dicho fichero, y no tengamos que compilar y desplegar otra vez.

Lo primero que hacemos es incluir, en el fichero web.xml, un nuevo parámetro de contexto con la ruta del servidor. A este parámetro lo hemos llamado rutaws:

```
<context-param>
    <param-name>rutaws</param-name>
    <param-value>https://10.1.8.1:8181/P1-ws-ws/VisaDAOWSService</param-
value>
```

CORRECCIÓN: hay que utilizar la url que no tiene http secure, es decir,
<http://10.1.8.1:8080/P1-ws-ws/VisaDAOWSService>

```
</context-param>
```

A continuación, en la función processRequest de procesaPago.java, obtenemos el valor de dicho parámetro, incluyendo la línea

```
String ruta = getServletContext().getInitParameter("rutaws");
```

y modificamos el objeto dao para introducir nosotros la ip, con las siguientes líneas:

```
BindingProvider bp = (BindingProvider) dao;
bp.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
```

ruta);

Este cambio se realiza también en cada una de las clases cliente que hace uso del servicio.

Ejercicio 10

En la clase delPagos.java añadimos los mismos tres import que añadimos para realizaPago, y eliminamos el import de VisaDAO:

```
import ssii2.visa.VisaDAOWSService; // Stub generado automáticamente  
import ssii2.visa.VisaDAOWS; // Stub generado automáticamente  
import javax.xml.ws.WebServiceRef;
```

A continuación, sustituimos la instancia de VisaDAO por la de VisaDAOWSService:

```
VisaDAOWSService service = new VisaDAOWSService();  
VisaDAOWS dao = service. GetVisaDAOWSPort ();
```

Respecto a los parámetros de entrada y al retorno de la función, delPagos() en el fichero visaDAOWS.java, ambos son tipos serializables (recibe un string y devuelve un int), así que no hay que hacer ni ninguna modificación: estos tipos de datos se traducen a xml sin ningún problema.

En la clase getPagos hacemos los mismos dos primeros cambios. Además, cambiamos la función getPagos para que devuelva un arraylist de pagobean en lugar de un array.

```
public ArrayList<PagoBean> getPagos(@WebParam(name = "idComercio") String  
idComercio)
```

Como el funcionamiento de la función consistía en almacenar los pagos en un arraylist y luego transformarlo en array y devolver el array, simplemente eliminamos este último paso y devolvemos el arraylist, este arraylist es el denominado **pagos** en la función.

Ejercicio 11

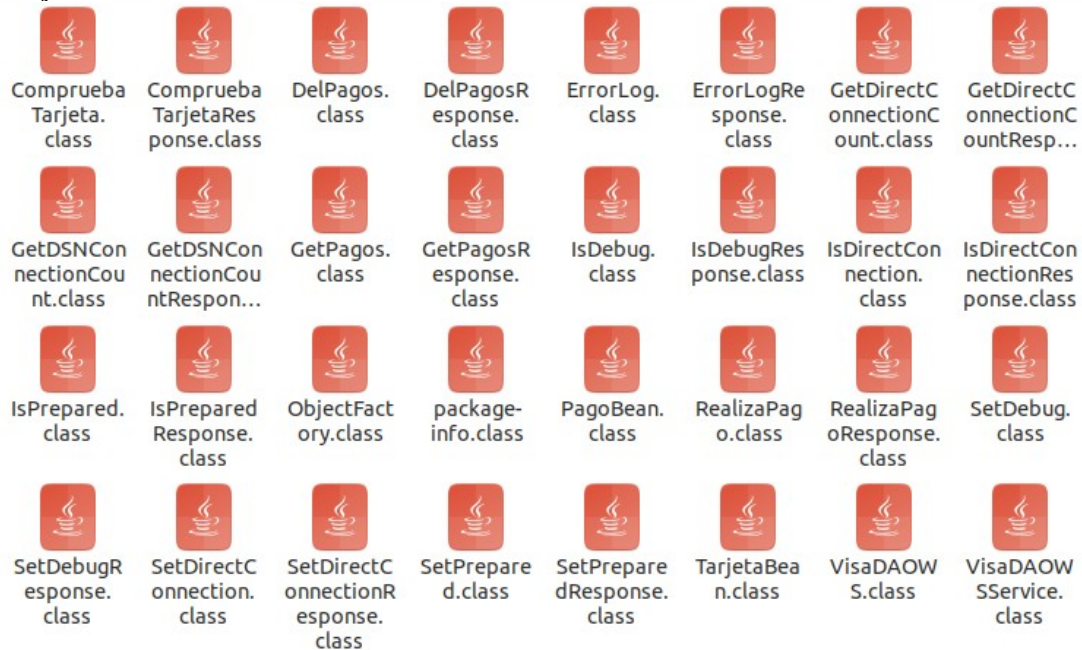
El comando que hemos tenido que utilizar es:

```
wsimport -d build/client/WEB-INF/classes -p ssii2.visa  
http://10.1.8.1:8080/P1-ws-ws/VisaDAOWSService?wsdl
```

siendo esta última url la que encontramos logeandonos como adminsitrador, y luego yendo a la aplicación P1-ws-ws y al Endpoint de VisaDAO.

Las clases generadas son las que se ven en la siguiente captura:

Victoria Pelayo Alvaredo
José Benjumedá Rubio



Todos estos archivos generados sirven para que el cliente conozca los métodos y las clases para comunicarse con el servidor.

Ejercicio 12

Incluimos en build.xml la llamada al comando wsimport con sus argumentos siguiendo el formato:

```
<exec executable="wsimport">
  <arg value="-d" />
  <arg value="${build.client}/WEB-INF/classes" />
  <arg value="-p" />
  <arg value="${paquete}" />
  <arg value="${wsdl.url}" />
</exec>
```

Ejercicio 13

Para realizar el despliegue en dos nodos, en build properties nos aseguramos de que las direcciones son las correctas, en nuestro caso:

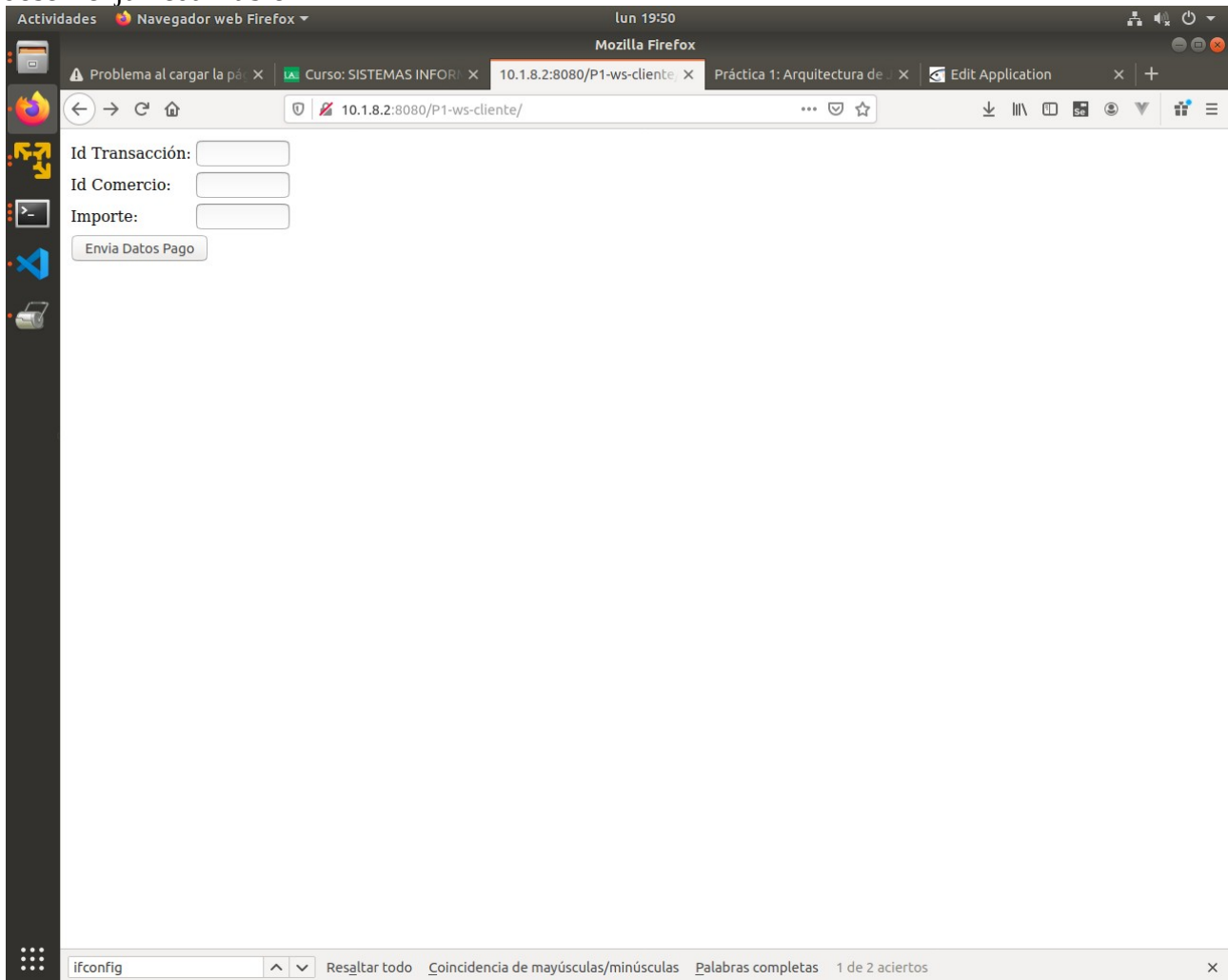
as.host.client=10.1.8.2

as.host.server=10.1.8.1

Posteriormente compilamos, empaquetamos y desplegamos el servidor. Y después generamos los stubs, compilamos, empaquetamos y desplegamos el cliente.

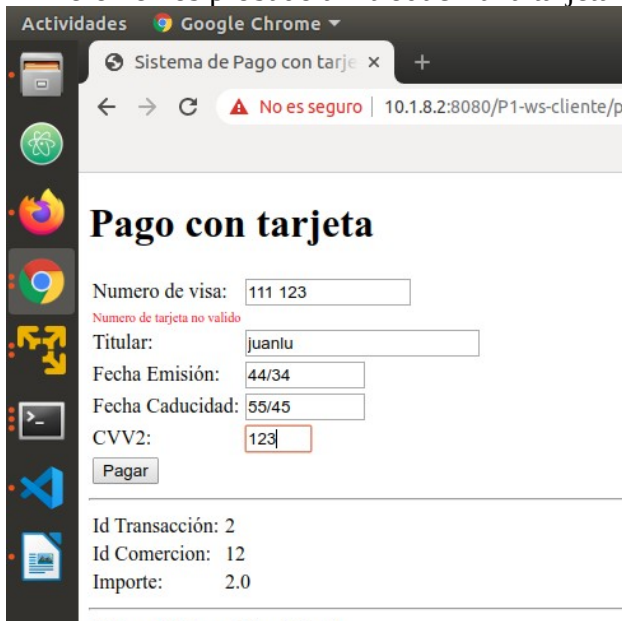
Nos metemos en la URL correspondiente y comprobamos que la parte de cliente ha funcionado:

Victoria Pelayo Alvaredo
José Benjumedá Rubio



Completamos los campos del primer formulario y nos pide los datos de la tarjeta.

Primero hemos probado a introducir una tarjeta cuyos datos no sean válidos :



Y nos devuelve una alerta de que los datos de la tarjeta introducida no son válidos

Victoria Pelayo Alvaredo
José Benjumeda Rubio



Actividades Google Chrome

Sistema de Pago con tarje x +

← → ↻ ⓘ No es seguro | 10.1.8.2:8080/P1-ws-c

Pago con tarjeta

Numero de visa:

Numero de tarjeta no valido

Titular:

Fecha Emisión:

Fecha Caducidad:

CVV2:

Pagar

Después probamos a introducir todos los datos correctamente, pero, nos da un error de tipo 500 que no hemos podido solucionar.

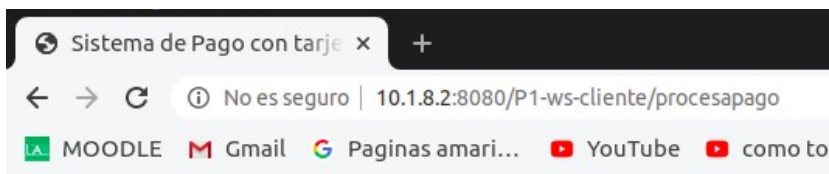
CORRECCIÓN: tras cambiar la url en la que teníamos https, además de una corrección de cuándo devolvemos el pago y cuando null en la función realizaPago. En este if:

```
if (!pstmt.execute())
    && pstmt.getUpdateCount() == 1) {
    ret = 1;
}
```

antes teníamos un ret=0, es decir, lo tomábamos como error en lugar de como éxito.

Estas son las únicas diferencias respecto a la entrega anterior.

Captura del ws funcionando:



Sistema de Pago con tarje x +

← → ↻ ⓘ No es seguro | 10.1.8.2:8080/P1-ws-cliente/procesapago

MOODLE Gmail Paginas amari... YouTube como to

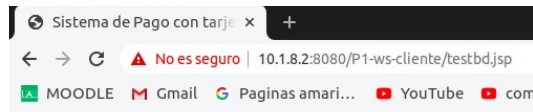
Pago con tarjeta

Pago realizado con éxito. A continuación se muestra el comprobante del mismo:

idTransaccion: 4
idComercio: 4
importe: 15.0
codRespuesta: 000
idAutorizacion: 9

[Volver al comercio](#)

Comprobación utilizando testbd.jsp de que el pago funciona:



Pago con tarjeta

Proceso de un pago

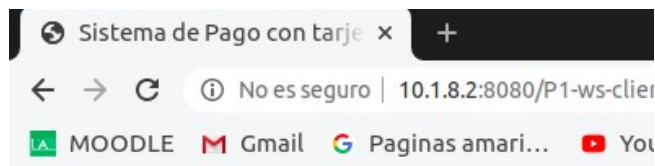
Id Transacción:
Id Comercio:
Importe:
Numero de visa:
Titular:
Fecha Emisión:
Fecha Caducidad:
CVV2:
Modo debug: ☐ True ☐ False
Direct Connection: ☐ True ☐ False
Use Prepared: ☐ True ☐ False

Consulta de pagos

Id Comercio:

Borrado de pagos

Id Comercio:



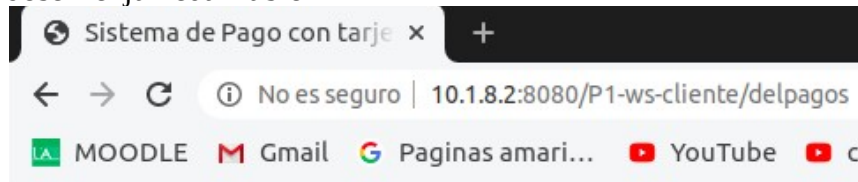
Pago con tarjeta

Lista de pagos del comercio 9

idTransaccion	Importe	codRespuesta	idAutorizacion
9	15.0	000	10

[Volver al comercio](#)

Victoria Pelayo Alvaredo
José Benjumedá Rubio



Pago con tarjeta

Se han borrado 1 pagos correctamente para el comercio 9

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

Cuestión 1

Primero **pago.html**, luego vas al servlet de **ComienzaPago**, después se pasa a **formdatosvisa.jsp**, de él vas al servlet de **ProcesaPago**, aquí es dónde se detectará que hay un error en la fecha de caducidad de la tarjeta. Este te retornará a **formdatosvisa.jsp** y de aquí se pasa a **error/muestraerror.jsp** que mostrará el error.

Cuestión 2

El servlet encargado de solicitar la información sobre la tarjeta es el de **ComienzaPago**.

Cuestión 3

El servlet de **ComienzaPago** es el que solicita los campos de la tarjeta que son: número de tarjeta, titular, fecha de emisión y fecha de caducidad. Se almacenan estos datos en una instancia de **PagoBean**, y se usa en **formdatosvisa.jsp** y en **ProcesaPago**.

Cuestión 4

Cuando se accede mediante “pago.html” no se introducen todos los datos directamente, si no, que se introduce primero id transacción, id comercio e importe. Y después se introducen los datos de la tarjeta, aquí se utiliza el servlet “ComienzaPago”.

Cuando se utiliza test.jsp se introducen todos los datos directamente.

En ambos casos se llama al servlet **ComienzaPago**, por eso ambos funcionan.