		Escuela Politécnica Superior Ingeniería Informática Prácticas de Sistemas Informáticos 2			
Grupo	2391	Práctica	1b	Fecha	27/02/2022
Alumno/a		Garcia Toledano, Carlos			
Alumno/a		Almarza Marques, Pablo			

Ejercicio número 1:

. Prepare e inicie una máquina virtual a partir de la plantilla si2srv con: 1GB de RAM asignada, 2 CPUs. A continuación:

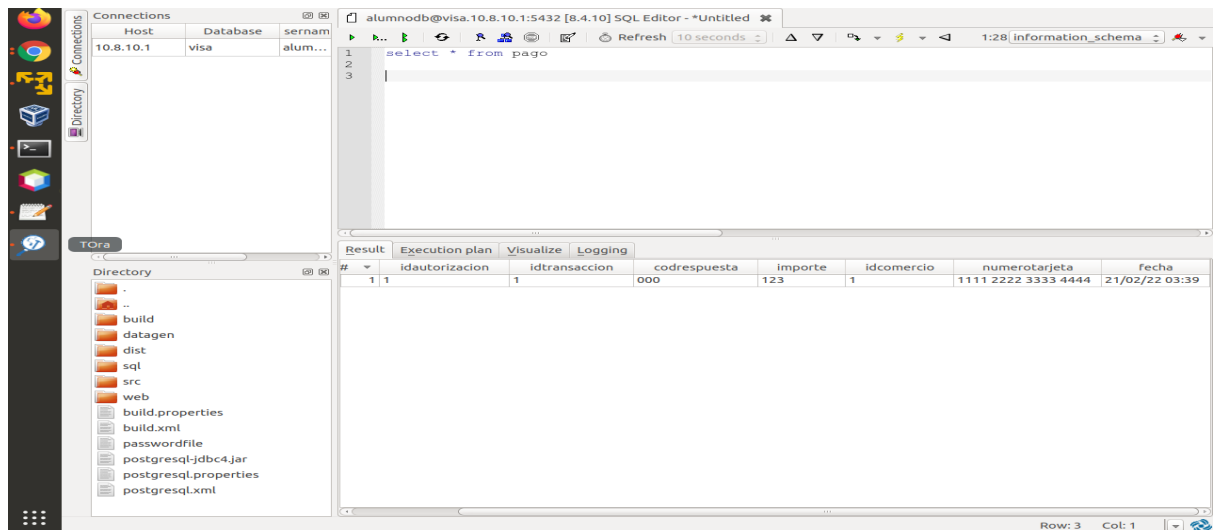
- Modifique los ficheros que considere necesarios en el proyecto para que se despliegue tanto la aplicación web como la base de datos contra la dirección asignada a la pareja de prácticas.
- Realice un pago contra la aplicación web empleando el navegador en la ruta <http://10.X.Y.Z:8080/P1> Conéctese a la base de datos (usando el cliente Tora por ejemplo) y obtenga evidencias de que el pago se ha realizado.
- Acceda a la página de pruebas extendida, <http://10.X.Y.Z:8080/P1/testbd.jsp>. Compruebe que la funcionalidad de listado de y borrado de pagos funciona correctamente. Elimine el pago anterior.

Primero se cambia el as.host, el db.client.host y el db.host para que sea el mismo que la IP de la máquina virtual. A continuación, se utiliza el comando ant todo para compilar y desplegar y crear la base de datos.

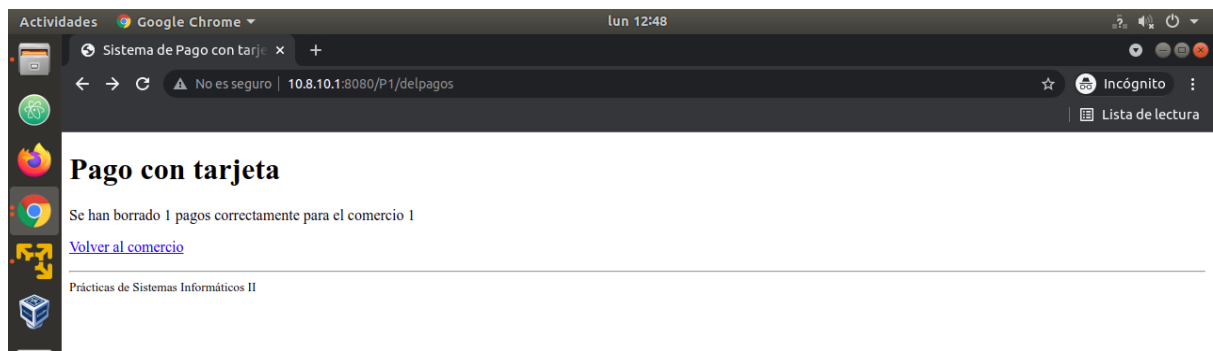
Se abre el servidor, y se despliega una página con Id transacción, Id Comercio y el importe. Introducimos unos datos y se despliega la página para pagar con tarjeta. Introducimos los datos de una de las tarjetas guardadas en la base de datos:

- numerotarjeta: 1111 2222 3333 4444
- titular: Jose Garcia
- validadesde: 11/09
- validahasta: 11/22
- codigoverificacion: 123

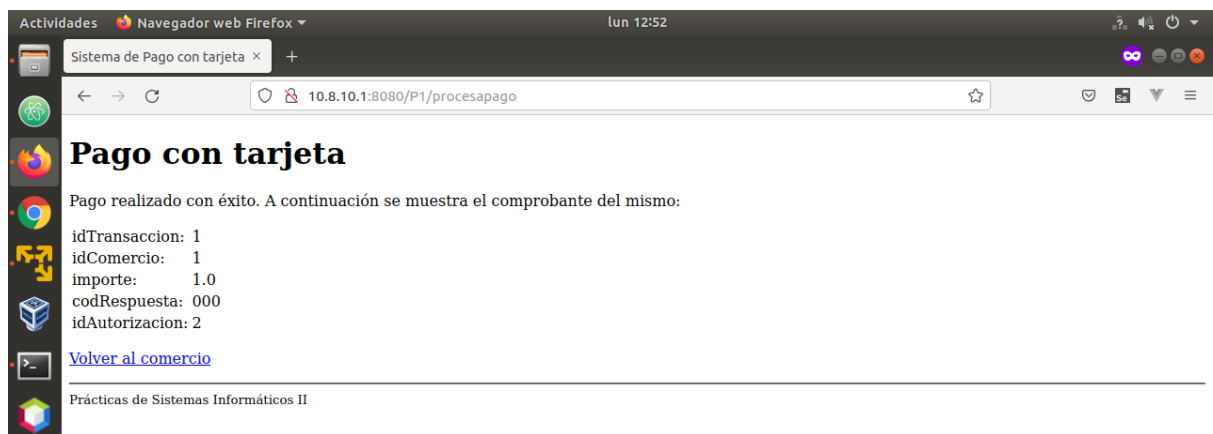
El servidor se encarga de procesar el pago comprobando la existencia de la tarjeta y la validez de los datos. Si todo está correcto, la transacción es guardada en la base de datos, como podemos comprobar en la siguiente captura de tora.



En la siguiente captura comprobamos que la funcionalidad de borrado de transacciones funciona correctamente.



En la siguiente, comprobamos que la funcionalidad de añadir transacciones funciona correctamente.



Ejercicio número 2:

La clase VisaDAO implementa los dos tipos de conexión descritos anteriormente, los cuales son heredados de la clase DBTester. Sin embargo, la configuración de la conexión utilizando la conexión directa es incorrecta. Se pide completar la información necesaria para llevar a cabo la conexión directa de forma correcta. Para ello habrá que fijar los atributos a los valores correctos. En particular, el nombre del driver JDBC a utilizar, el JDBC connection string que se debe corresponder con el servidor postgresql, y el nombre de usuario y la contraseña. Es necesario consultar el apéndice 10 para ver los detalles de cómo se obtiene una conexión de forma correcta. Una vez completada la información, acceda a la página de pruebas extendida, <http://10.X.Y.Z:8080/P1/testbd.jsp> y pruebe a realizar un pago utilizando la conexión directa y pruebe a listarlo y eliminarlo. Adjunte en la memoria evidencias de este proceso, incluyendo capturas de pantalla

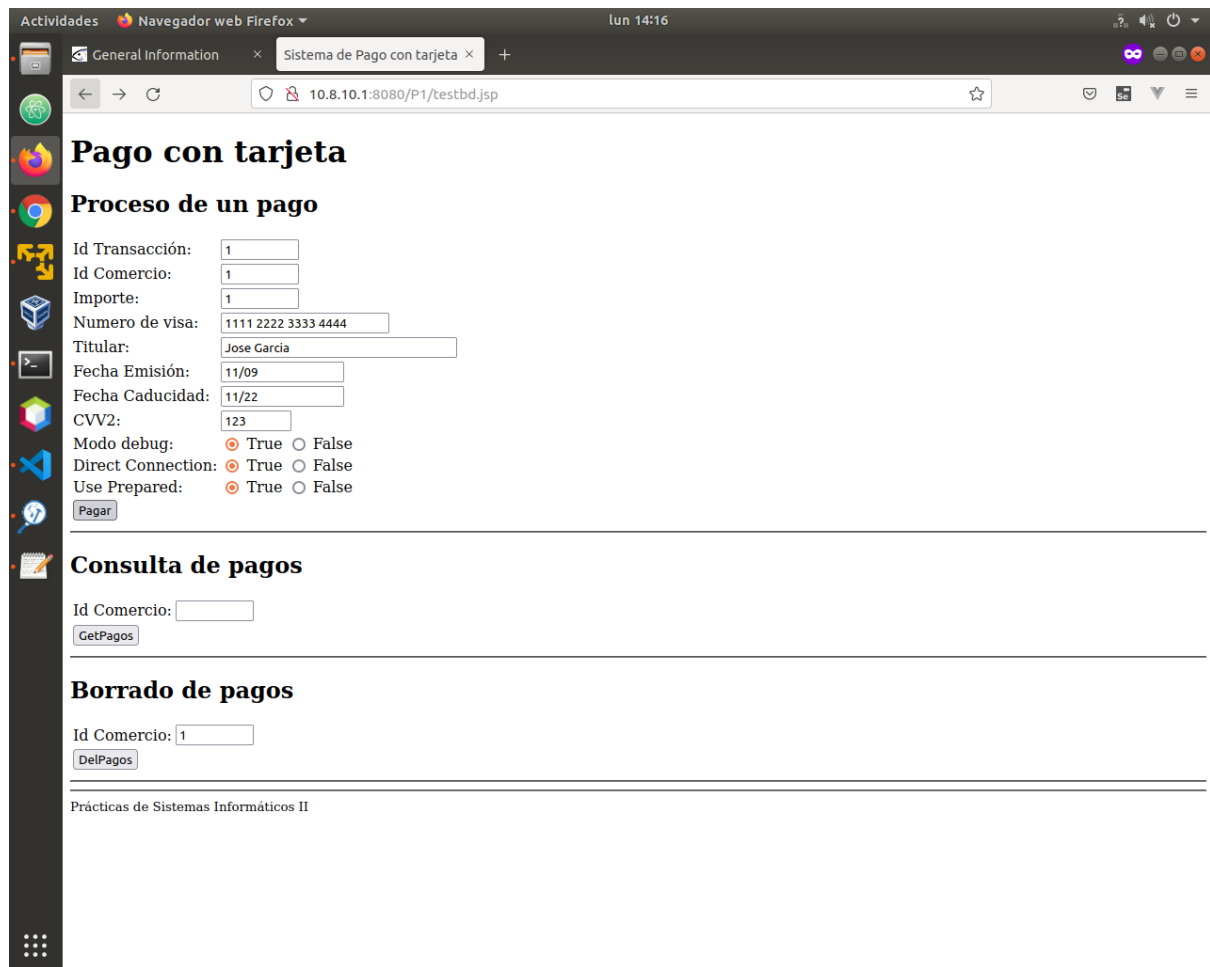
Para completar este ejercicio solo tenemos que cambiar unas variables en el fichero DBTester.

```
private static final String JDBC_DRIVER =
    "org.postgresql.Driver";

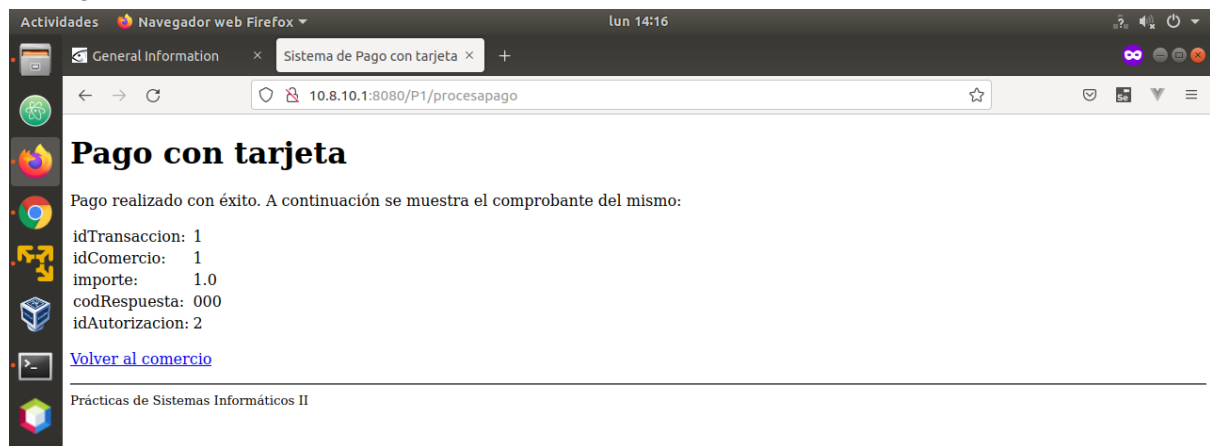
// TODO: Definir la cadena de conexión a la base de datos
/*****/
private static final String JDBC_CONNSTRING =
    "jdbc:postgresql://10.8.10.1:5432/visa";
/*****/
private static final String JDBC_USER = "alumnodb";
private static final String JDBC_PASSWORD = "****";
```

Estos datos los obtenemos del fichero postgresql.properties.

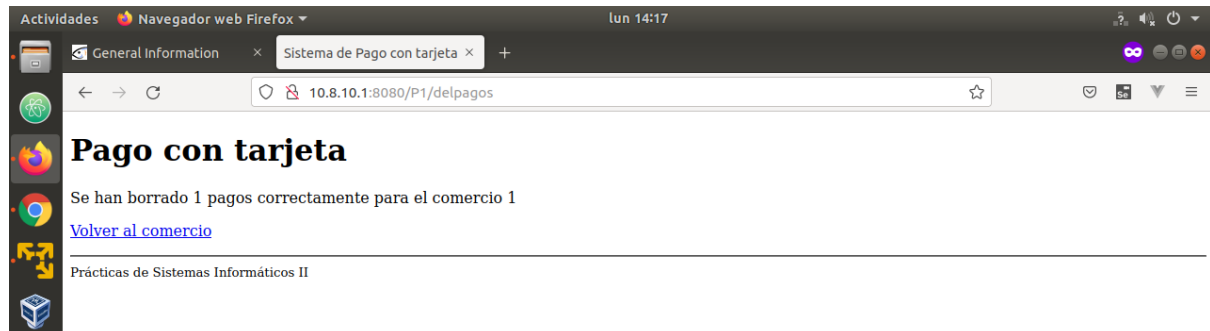
Primera captura nos muestra cómo introducimos datos para una nueva transacción usando conexión directa.



La segunda captura nos muestra la transacción realizada con éxito.



La tercera, nos muestra el borrado de la transacción realizado con éxito.



Ejercicio número 3:

Examinar el archivo `postgresql.properties` para determinar el nombre del recurso JDBC correspondiente al `DataSource` y el nombre del pool. Acceda a la Consola de Administración. Compruebe que los recursos JDBC y pool de conexiones han sido correctamente creados. Realice un Ping JDBC a la base de datos. Anote en la memoria de la práctica los valores para los parámetros Initial and Minimum Pool Size, Maximum Pool Size, Pool Resize Quantity, Idle Timeout, Max Wait Time. Comente razonadamente qué impacto considera que pueden tener estos parámetros en el rendimiento de la aplicación.

Ping:	<input checked="" type="checkbox"/> Enabled	When enabled, the pool is pinged during creation or reconfiguration to identify and warn of any erroneous values for its attributes
Deployment Order:	<input type="text" value="100"/>	Specifies the loading order of the resource at server startup. Lower numbers are loaded first.
Description:	<input type="text"/>	

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

Aumentar el pool size inicial hace que el server tarde más en arrancar ya que tiene que crear todos los threads, pero luego tarda menos en responder a las peticiones ya que ya tiene creados los threads.

Aumentar el pool size máximo hace que se puedan responder a más llamadas pero esto tiene un coste en el server que utilizará más recursos.

Aumentar el pool resize hace que cuando hay que hacer un resize del pool, a este le cueste más hacerlo, pero lo tendrá que hacer menos veces ya que pasará más tiempo hasta que tenga que aumentar el tamaño de nuevo en el caso de que haga falta.

Aumentar el idle timeout hace que una conexión pueda estar más tiempo sin hacer nada sin ser expulsada del hilo. Esto puede interesarnos en el caso de que por esa conexión se vuelve a transmitir información al rato, nos ahorraríamos cerrar y abrir esa conexión.

Ejercicio número 4:

Localice los siguientes fragmentos de código SQL dentro del proyecto proporcionado (P1-base) correspondientes a los siguientes procedimientos:

- Consulta de si una tarjeta es válida.
- Ejecución del pago.

-Consulta de si una tarjeta es válida

```
String getQryCompruebaTarjeta(TarjetaBean tarjeta) {  
    String qry = "select * from tarjeta "  
        + "where numeroTarjeta='" + tarjeta.getNumero()  
        + "' and titular='" + tarjeta.getTitular()  
        + "' and validaDesde='" + tarjeta.getFechaEmision()  
        + "' and validaHasta='" +  
tarjeta.getFechaCaducidad()  
        + "' and codigoVerificacion='" +  
tarjeta.getCodigoVerificacion() + "'";  
    return qry;  
}
```

-Ejecución del pago

```
String getQryInsertPago(PagoBean pago) {  
    String qry = "insert into pago("  
        + "idTransaccion,"  
        + "importe,idComercio,"  
        + "numeroTarjeta)"  
        + " values ("  
        + "'" + pago.getIdTransaccion() + "',"  
        + pago.getImporte() + ","  
        + "'" + pago.getIdComercio() + "',"  
        + "'" + pago.getTarjeta().getNumero() + "'"  
        + ")";  
    return qry;  
}
```

Ejercicio número 5:

Edite el fichero VisaDAO.java y localice el método errorLog. Compruebe en qué partes del código se escribe en log utilizando dicho método. Realice un pago utilizando la página testbd.jsp con la opción de debug activada. Visualice el log del servidor de aplicaciones y compruebe que dicho log contiene información adicional sobre las acciones llevadas a cabo en VisaDAO.java.

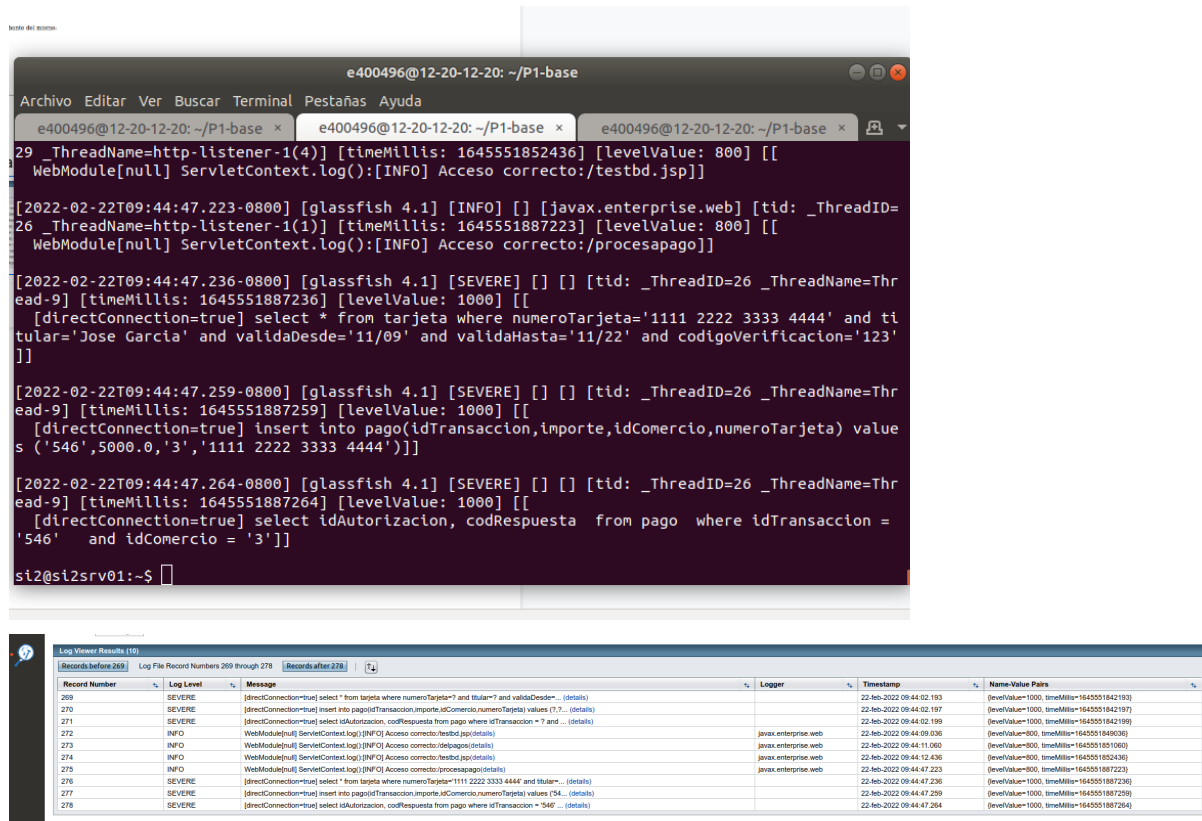
```
public void errorLog(String error) {  
    if (isDebug())  
        System.err.println("[directConnection=" +  
this.isDirectConnection() + "] " +  
        error);  
}
```

errorLog() es usado en muchas partes del código, sobre todo en VisaDAO.java para mantener un log que nos de información de lo que hace el server.

Estos son los datos del pago que hemos realizado para observar el log:



Y aquí está el log en la terminal del server y en la consola de glassfish:



The terminal window shows the following log output:

```
e400496@12-20-12-20: ~/P1-base
Archivo Editar Ver Buscar Terminal Pestañas Ayuda
e400496@12-20-12-20: ~/P1-base x e400496@12-20-12-20: ~/P1-base x e400496@12-20-12-20: ~/P1-base x
29 _ThreadName=http-listener-1(4)] [timeMillis: 1645551852436] [levelValue: 800] [[
WebModule[null] ServletContext.log():[INFO] Acceso correcto:/testbd.jsp]]

[2022-02-22T09:44:47.223-0800] [glassfish 4.1] [INFO] [] [javax.enterprise.web] [tid: _ThreadID=
26 _ThreadName=http-listener-1(1)] [timeMillis: 1645551887223] [levelValue: 800] [[
WebModule[null] ServletContext.log():[INFO] Acceso correcto:/procesapago]]

[2022-02-22T09:44:47.236-0800] [glassfish 4.1] [SEVERE] [] [] [tid: _ThreadID=26 _ThreadName=Thr
ead-9] [timeMillis: 1645551887236] [levelValue: 1000] [[
[directConnection=true] select * from tarjeta where numeroTarjeta='1111 2222 3333 4444' and ti
tular='Jose Garcia' and validaDesde='11/09' and validaHasta='11/22' and codigoVerificacion='123'
]]

[2022-02-22T09:44:47.259-0800] [glassfish 4.1] [SEVERE] [] [] [tid: _ThreadID=26 _ThreadName=Thr
ead-9] [timeMillis: 1645551887259] [levelValue: 1000] [[
[directConnection=true] insert into pago(idTransaccion,importe,idComercio,numeroTarjeta) value
s ('546',5000.0,'3','1111 2222 3333 4444')]]

[2022-02-22T09:44:47.264-0800] [glassfish 4.1] [SEVERE] [] [] [tid: _ThreadID=26 _ThreadName=Thr
ead-9] [timeMillis: 1645551887264] [levelValue: 1000] [[
[directConnection=true] select idAutorizacion, codRespuesta from pago where idTransaccion =
'546' and idComercio = '3']]

si2@si2srv01:~$
```

The Log Viewer window shows the following table:

Record Number	Log Level	Message	Logger	Timestamp	Name-Value Pairs
269	SEVERE	[directConnection=true] select * from tarjeta where numeroTarjeta='1111 2222 3333 4444' and titular='Jose Garcia' and validaDesde='11/09' and validaHasta='11/22' and codigoVerificacion='123'	javax.enterprise.web	22-Feb-2022 09:44:02.193	[levelValue=1000, timeMillis=1645551842193]
270	SEVERE	[directConnection=true] insert into pago(idTransaccion,importe,idComercio,numeroTarjeta) values (7,5000.0,'3','1111 2222 3333 4444')	javax.enterprise.web	22-Feb-2022 09:44:02.197	[levelValue=1000, timeMillis=1645551842197]
271	SEVERE	[directConnection=true] select idAutorizacion, codRespuesta from pago where idTransaccion = '7' and idComercio = '3'	javax.enterprise.web	22-Feb-2022 09:44:02.199	[levelValue=1000, timeMillis=1645551842199]
272	INFO	WebModule[null] ServletContext.log():[INFO] Acceso correcto:/testbd.jsp	javax.enterprise.web	22-Feb-2022 09:44:09.036	[levelValue=800, timeMillis=1645551849036]
273	INFO	WebModule[null] ServletContext.log():[INFO] Acceso correcto:/procesapago	javax.enterprise.web	22-Feb-2022 09:44:11.060	[levelValue=800, timeMillis=1645551851060]
274	INFO	WebModule[null] ServletContext.log():[INFO] Acceso correcto:/procesapago	javax.enterprise.web	22-Feb-2022 09:44:12.436	[levelValue=800, timeMillis=1645551852436]
275	INFO	WebModule[null] ServletContext.log():[INFO] Acceso correcto:/procesapago	javax.enterprise.web	22-Feb-2022 09:44:12.436	[levelValue=800, timeMillis=1645551852436]
276	SEVERE	[directConnection=true] select * from tarjeta where numeroTarjeta='1111 2222 3333 4444' and titular='Jose Garcia' and validaDesde='11/09' and validaHasta='11/22' and codigoVerificacion='123'	javax.enterprise.web	22-Feb-2022 09:44:47.223	[levelValue=1000, timeMillis=1645551887223]
277	SEVERE	[directConnection=true] insert into pago(idTransaccion,importe,idComercio,numeroTarjeta) values ('546',5000.0,'3','1111 2222 3333 4444')	javax.enterprise.web	22-Feb-2022 09:44:47.259	[levelValue=1000, timeMillis=1645551887259]
278	SEVERE	[directConnection=true] select idAutorizacion, codRespuesta from pago where idTransaccion = '546' and idComercio = '3'	javax.enterprise.web	22-Feb-2022 09:44:47.264	[levelValue=1000, timeMillis=1645551887264]

Podemos observar en las anteriores transacciones del logfile que si no está activado el modo debug, no aparecen los datos de la tarjeta.

Ejercicio número 6:

Realicé las modificaciones necesarias en VisaDAOWS.java para que implemente de manera correcta un servicio web. Los siguientes métodos y todos sus parámetros deberán ser publicados como métodos del servicio.

compruebaTarjeta()

realizaPago()

isDebug() / setDebug() (Nota: VisaDAO.java contiene dos métodos setDebug que reciben distintos argumentos. Solo uno de ellos podrá ser exportado como servicio web)3 .

isPrepared() / setPrepared()

Deberemos publicar así mismo:

isDirectConnection() / setDirectConnection() que son métodos heredados de la clase DBTester.

Para ello, implemente estos métodos también en la clase hija. Es decir, haga un override de Java, implementando estos métodos en VisaDAOWS mediante invocaciones a la clase padre (super). En ningún caso se debe añadir ni modificar nada de la clase DBTester.

Modifique así mismo el método realizaPago() para que éste devuelva el pago modificado tras la correcta o incorrecta realización del pago:

Con identificador de autorización y código de respuesta correcto en caso de haberse realizado.

Con null en caso de no haberse podido realizar.

¿Por qué se ha de alterar el parámetro de retorno del método realizaPago() para que devuelva

el pago el lugar de un boolean?

Primero hacemos que el parámetro a retornar sea del tipo PagoBean.

```
PagoBean ret = null;
```

Después en la comprobación de isPrepared(), seteamos el return a pago en caso correcto y a null en caso de error.

```
if (isPrepared() == true) {
    String insert = INSERT_PAGOS_QRY;
    errorLog(insert);
    pstmt = con.prepareStatement(insert);
    pstmt.setString(1, pago.getIdTransaccion());
    pstmt.setDouble(2, pago.getImporte());
    pstmt.setString(3, pago.getIdComercio());
    pstmt.setString(4, pago.getTarjeta().getNumero());
    ret = null;
    if (!pstmt.execute()
        && pstmt.getUpdateCount() == 1) {
        ret = pago;
    }
} else {
    stmt = con.createStatement();
    String insert = getQryInsertPago(pago);
    errorLog(insert);
    ret = null;
    if (!stmt.execute(insert)
        && stmt.getUpdateCount() == 1) {
        ret = pago;
    }
}
```

Seguimos con la comprobación de rs.next() en la que ponemos el return a null en caso de error de nuevo

```
if (rs.next()) {
    pago.setIdAutorizacion(String.valueOf(rs.getInt("idAutorizacion")));
    pago.setCodRespuesta(rs.getString("codRespuesta"));
} else {
    ret = null;
}
```

Finalmente lo volvemos a poner como null en caso de que suceda alguna excepción

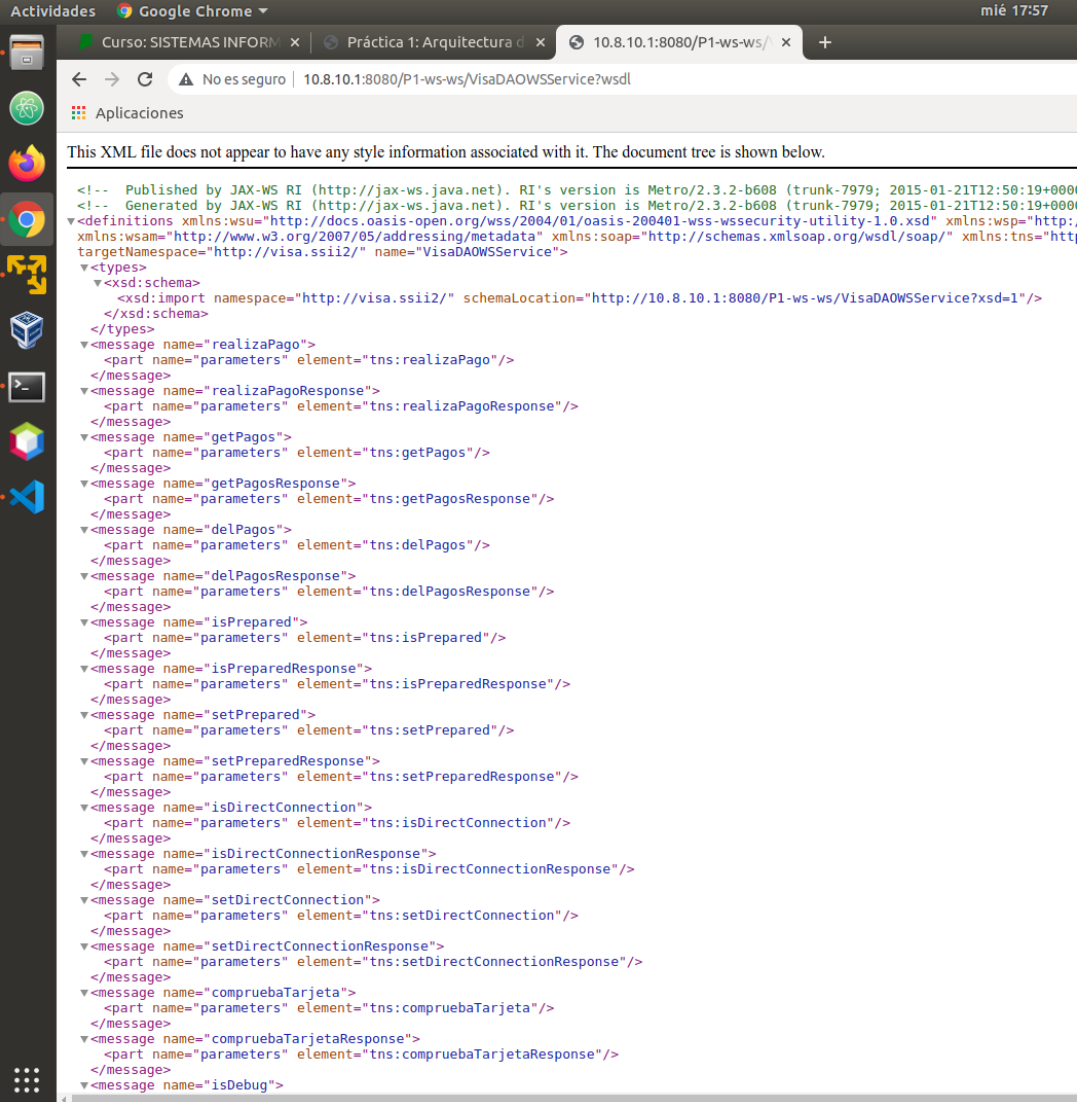
```
catch (Exception e) {
    errorLog(e.toString());
    ret = null;
}
```

Se debe alterar el retorno para poder pasar la información del pago.

Ejercicio número 7:

Despliegue el servicio con la regla correspondiente en el build.xml. Acceda al WSDL remotamente con el navegador e inclúyalo en la memoria de la práctica (habrá que asegurarse que la URL contiene la dirección IP de la máquina virtual donde se encuentra el servidor de aplicaciones).

Hay que cambiar la URL y poner la IP de la máquina virtual para acceder al fichero WSDL (en Mozilla Firefox no funciona).



```
<!-- 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)
-->
<!-- 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)
-->
<?xml version='1.0'?>
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns:wsp="http://
xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://
targetNamespace="http://visa.ssii2/" name="VisaDAOWSService">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://visa.ssii2/" schemaLocation="http://10.8.10.1:8080/P1-ws-ws/VisaDAOWSService?xsd=1"/>
    </xsd:schema>
  </types>
  <message name="realizaPago">
    <part name="parameters" element="tns:realizaPago"/>
  </message>
  <message name="realizaPagoResponse">
    <part name="parameters" element="tns:realizaPagoResponse"/>
  </message>
  <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="isDirectConnection">
    <part name="parameters" element="tns:isDirectConnection"/>
  </message>
  <message name="isDirectConnectionResponse">
    <part name="parameters" element="tns:isDirectConnectionResponse"/>
  </message>
  <message name="setDirectConnection">
    <part name="parameters" element="tns:setDirectConnection"/>
  </message>
  <message name="setDirectConnectionResponse">
    <part name="parameters" element="tns:setDirectConnectionResponse"/>
  </message>
  <message name="compruebaTarjeta">
    <part name="parameters" element="tns:compruebaTarjeta"/>
  </message>
  <message name="compruebaTarjetaResponse">
    <part name="parameters" element="tns:compruebaTarjetaResponse"/>
  </message>
  <message name="isDebug">
```

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

```
xsd:import namespace="http://visa.ssii2/"
schemaLocation="http://10.8.10.1:8080/P1-ws-ws/VisaDAOWSService?xsd=1/"
```

-¿Qué tipos de datos predefinidos se usan?

En el fichero podemos observar string, boolean, int y double. Los identificamos porque les sigue un xs

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

TarjetaBean y PagoBean. Estos siguen a la etiqueta tns.

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

<messages>

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

<operation>

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

<binding>

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

<service>

Ejercicio número 8:

Realícese las modificaciones necesarias en `ProcesaPago.java` para que implemente de manera correcta la llamada al servicio web mediante stubs estáticos. Téngase en cuenta que:

El nuevo método `realizaPago()` ahora no devuelve un boolean, sino el propio objeto `Pago` modificado.

Las llamadas remotas pueden generar nuevas excepciones que deberán ser tratadas en el código cliente.

Después de hacer los imports indicados, creamos el nuevo `VisaDAOWS` como se nos indica en la función `processRequest()` del fichero `ProcesaPago.java`. Además, metemos todas las llamadas a la variable `dao` dentro de un `try` ya que ahora pueden lanzar excepciones.

```
try{

    VisaDAOWSService service = new VisaDAOWSService();
    VisaDAOWS dao = service.getVisaDAOWSPort();

    HttpSession sesion = request.getSession(false);
    if (sesion != null) {
        pago = (PagoBean)
sesion.getAttribute(ComienzaPago.ATTR_PAGO);
    }
    if (pago == null) {
```

```

        pago = creaPago(request);
        boolean isdebug =
Boolean.valueOf(request.getParameter("debug"));
        dao.setDebug(isdebug);
        boolean isdirectConnection =
Boolean.valueOf(request.getParameter("directConnection"));
        dao.setDirectConnection(isdirectConnection);
        boolean usePrepared =
Boolean.valueOf(request.getParameter("usePrepared"));
        dao.setPrepared(usePrepared);
    }

    // Almacenamos la tarjeta en el pago
    pago.setTarjeta(tarjeta);

    if (! dao.compruebaTarjeta(tarjeta)) {
        enviaError(new Exception("Tarjeta no autorizada:"),
request, response);
        return;
    }

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

    request.setAttribute(ComienzaPago.ATTR_PAGO, pago);
    if (sesion != null) sesion.invalidate();
    reenvia("/pagoexito.jsp", request, response);
    return;
} catch (Exception e) {
    System.err.println("ERROR: Error processing request.\n");
}
}

```

Como habíamos cambiado el return de realizaPago(), ahora hay que cambiar como se handlea dicha función:

```

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

```

Ejercicio número 9:

Modifique la llamada al servicio para que la ruta (URL) al servicio remoto se obtenga del fichero de configuración web.xml. Para saber cómo hacerlo consulte el apéndice 15.1 para más información y edite el fichero web.xml y analice los comentarios que allí se incluyen.

Para acceder al URL desde el XML, realizamos una llamada al servlet context usando el parámetro que creamos en el web.xml.

ProcesaPago.java:

```
BindingProvider bp = (BindingProvider) dao;

bp.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
getServletContext().getInitParameter("URLVisaDAOWSService"));
```

web.xml:

```
<context-param>
<param-name>URLVisaDAOWSService</param-name>

<param-value>http://10.8.10.1:8080/P1-ws-ws/VisaDAOWSService</param-value>
</context-param>
```

Ejercicio número 10:

Siguiendo el patrón de los cambios anteriores, adaptar las siguientes clases cliente para que toda la funcionalidad de la página de pruebas testbd.jsp se realice a través del servicio web. Esto afecta al menos a los siguientes recursos:

Servlet DelPagos.java: la operación dao.delPagos() debe implementarse en el servicio web.

Servlet GetPagos.java: la operación dao.getPagos() debe implementarse en el servicio web.

Tenga en cuenta que no todos los tipos de datos son compatibles con JAXB (especifica como codificar clases java como documentos XML), por lo que es posible que tenga que modificar el valor de retorno de alguno de estos métodos. Los apéndices contienen más información. Más específicamente, se tiene que modificar la declaración actual del método getPagos(), que devuelve un PagoBean[], por:

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

Hay que tener en cuenta que la página listapagos.jsp espera recibir un array del tipo PagoBean[]. Por ello, es conveniente, una vez obtenida la respuesta, convertir el ArrayList a un array de tipo PagoBean[] utilizando el método toArray() de la clase ArrayList.

Cambiamos el return de la función getPagos() por un ArrayList de tipo PagoBean, y consecuentemente cambiamos el return para que también sea un arraylist.

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

```
pagos = new ArrayList<PagoBean>();
```

```

        while (rs.next()) {
            TarjetaBean t = new TarjetaBean();
            PagoBean p = new PagoBean();
            p.setIdTransaccion(rs.getString("idTransaccion"));
            p.setIdComercio(rs.getString("idComercio"));
            p.setImporte(rs.getFloat("importe"));
            t.setNumero(rs.getString("numeroTarjeta"));
            p.setTarjeta(t);
            p.setCodRespuesta(rs.getString("codRespuesta"));

p.setIdAutorizacion(String.valueOf(rs.getInt("idAutorizacion")));

            pagos.add(p);
        }

        ret = pagos;

```

Después de cambiar el return de getPagos(), y como queremos seguir pasando un array a setAttribute(), hacemos las siguientes líneas de código:

```

ArrayList<PagoBean> pagos = dao.getPagos(idComercio);

PagoBean arr[] = new PagoBean[pagos.size()];
arr = pagos.toArray(arr);

request.setAttribute(ATTR_PAGOS, arr);

```

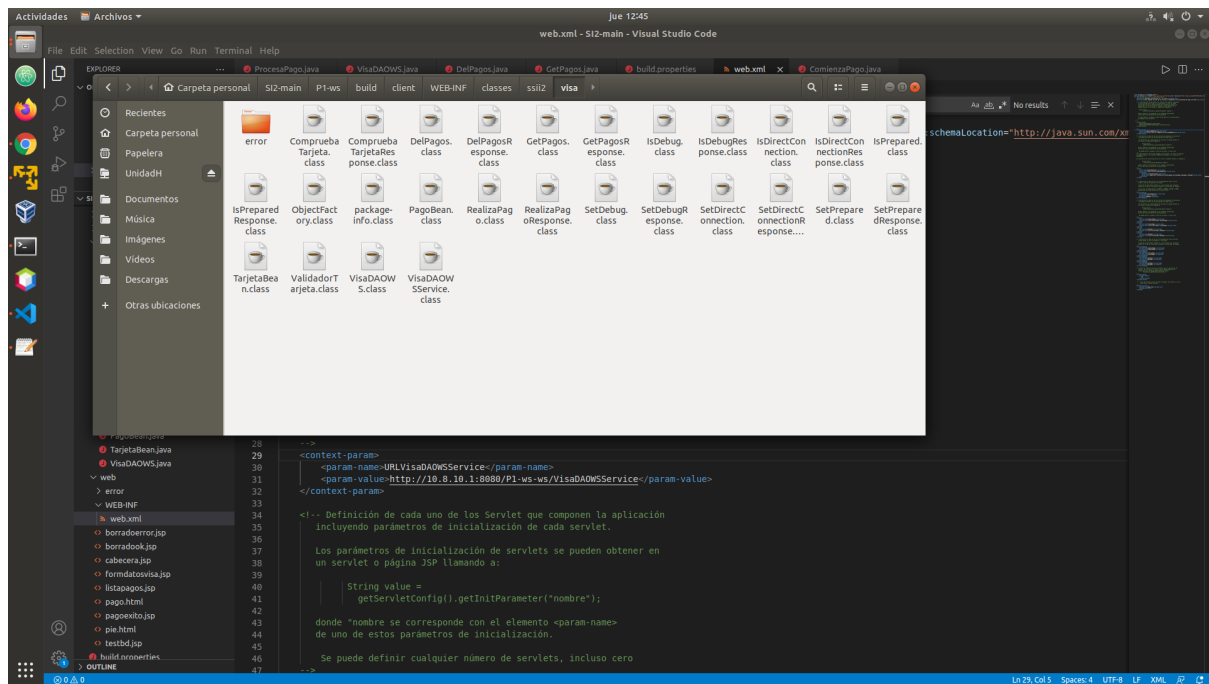
Ejercicio número 11:

Realice una importación manual del WSDL del servicio sobre el directorio de clases local. Anote en la memoria qué comando ha sido necesario ejecutar en la línea de comandos, qué clases han sido generadas y por qué. Téngase en cuenta que el servicio debe estar previamente desplegado.

Teniendo el servidor desplegado, utilizamos el comando

wsimport -d build/client/WEB-INF/classes -p ssii2.visa

<http://10.8.10.1:8080/P1-ws-ws/VisaDAOWSService?wsdl> dando lo siguiente:



Ejercicio número 12:

Complete el target generar-stubs definido en build.xml para que invoque a wsimport (utilizar la funcionalidad de ant exec para ejecutar aplicaciones). Téngase en cuenta que:

- El raíz del directorio de salida del compilador para la parte cliente ya está definido en build.properties como `${build.client}/WEB-INF/classes`
- El paquete Java raíz (ssii2) ya está definido como `${paquete}`
- La URL ya está definida como `${wsdl.url}`

Añadimos las siguientes líneas de código en el web.xml para ejecutar el wsimport cuando ejecutamos ant generar-stubs.

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

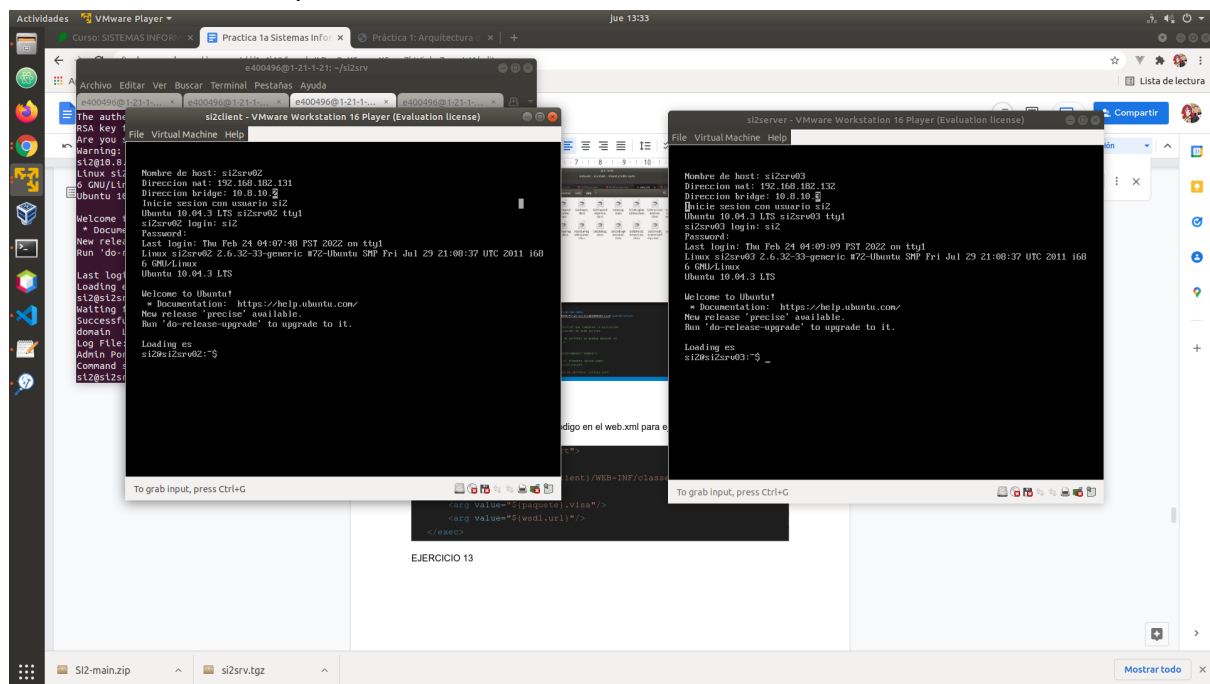
Ejercicio número 13:

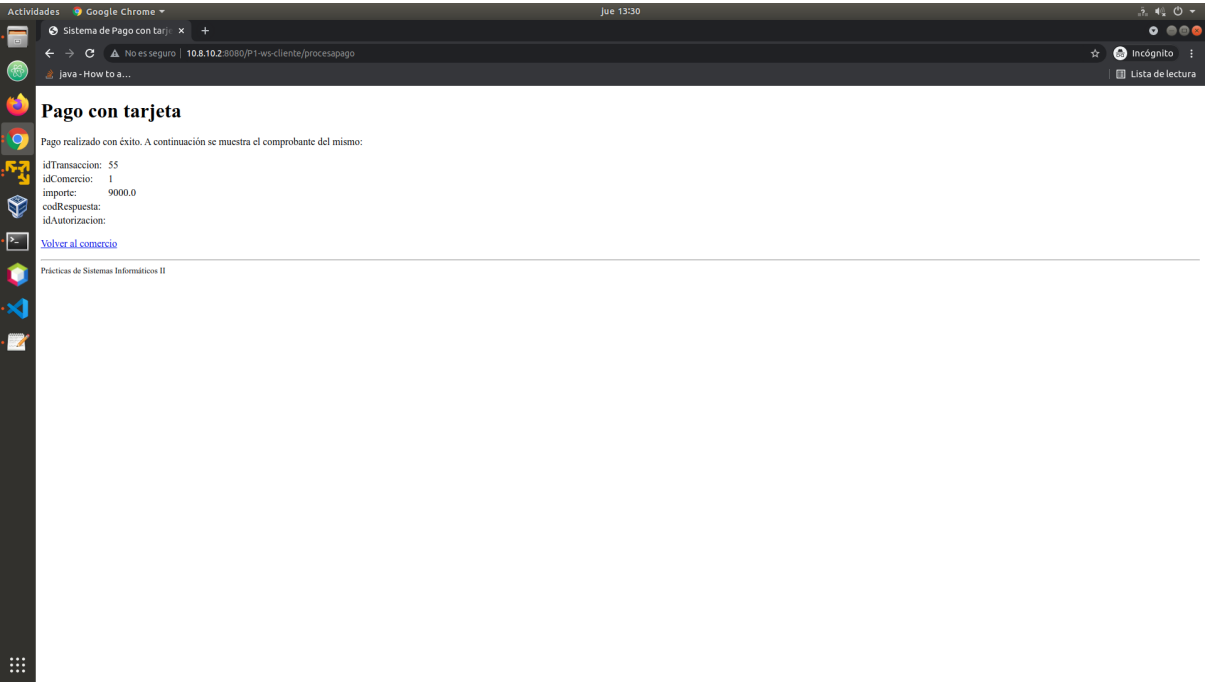
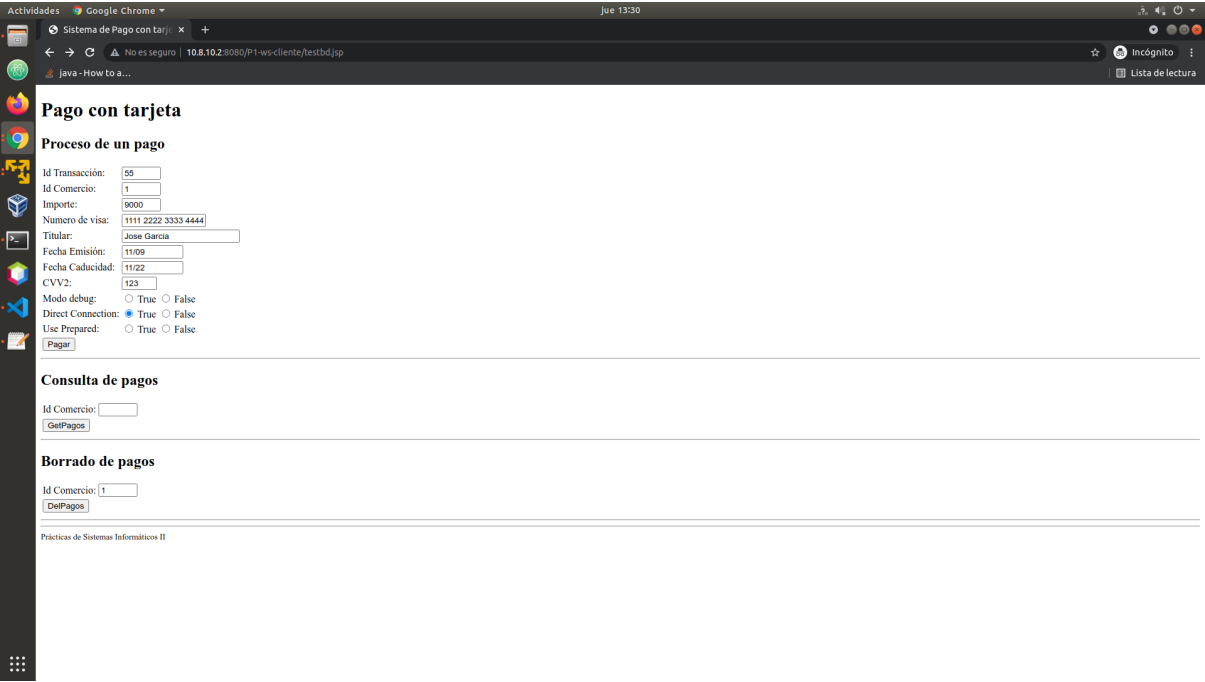
Realice un despliegue de la aplicación completo en dos nodos tal y como se explica en la Figura 8. Habrá que tener en cuenta que ahora en el fichero `build.properties` hay que especificar la dirección IP del servidor de aplicaciones donde se desplegará la parte del cliente de la aplicación y la dirección IP del servidor de aplicaciones donde se desplegará la parte del servidor. Las variables `as.host.client` y `as.host.server` deberán contener esta información.

Probar a realizar pagos correctos a través de la página testbd.jsp. Ejecutar las consultas SQL necesarias para comprobar que se realiza el pago. Anotar en la memoria práctica los resultados en forma de consulta SQL y resultados sobre la tabla de pagos.

(Texto de la respuesta)

Para este ejercicio, cambiamos las direcciones IP del cliente y el server que antes era la misma, ya que ahora tenemos 2 máquinas virtuales con diferentes IPs. En nuestro caso la IP del cliente es 10.8.10.2 y la del server 10.8.10.3, teniendo en cuenta que la base de datos reside en la máquina virtual del server.





Actividades

Tora

Jue 13:31

File Edit View Tools Editor Window Help

alumnodb@visa.10.8.10.3:5432 [8.4.10]

Connections

Host Database sernam
10.8.10.3 visa alum...

alumnodb@visa.10.8.10.3:5432 [8.4.10] SQL Editor - *Untitled

information_schema

1 select * from pago
2

Refresh 10 seconds

Directory

Result Execution plan Visualize Logging

Idautorizacion Idtransaccion codrespuesta importe Idcomercio numerotarjeta fecha
1,2 55 000 9000 1 1111 2222 3333 4444 24/02/22 04:30

Row: 2 Col: 1

GetPagos

Borrado de pagos

Id Comercio:

DelPagos

Prácticas de Sistemas Informáticos II

Actividades

Google Chrome

Jue 13:32

Sistema de Pago con tarj Nueva pestaña

No es seguro | 10.8.10.2:3080/P1-ws-cliente/procesapago

Haz clic para retroceder una página o pulsa unos segundos para ver el historial

Incógnito

Lista de lectura

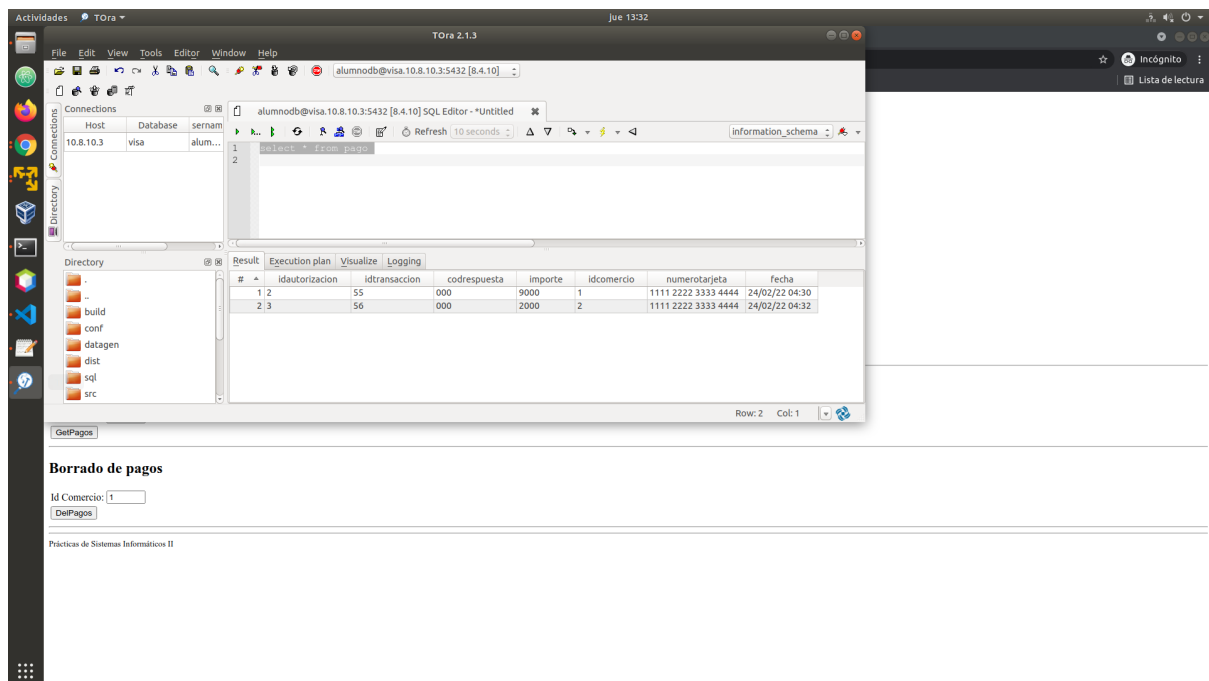
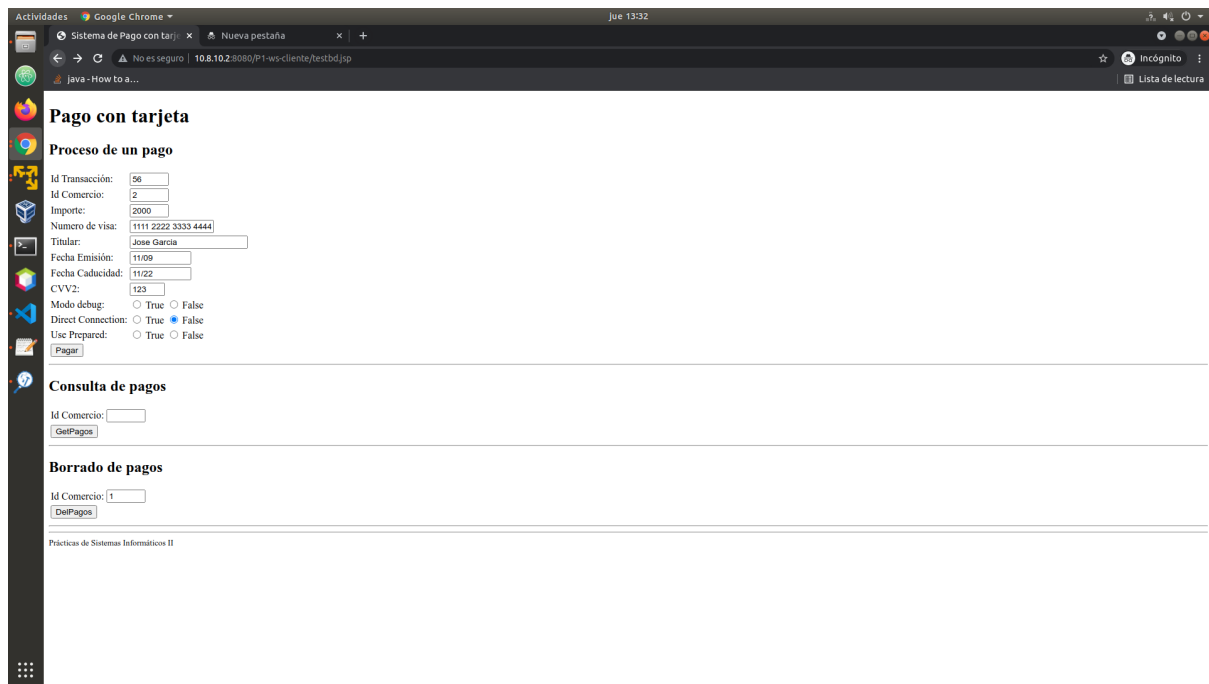
Pago con tarjeta

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

idTransaccion: 56
idComercio: 2
importe: 2000.0
codRespuesta:
idAutorizacion:

Volver al comercio

Prácticas de Sistemas Informáticos II



Cuestión número 1:

Teniendo en cuenta el diagrama de la Figura 3, indicar las páginas html, jsp y servlets por los que se pasa para realizar un pago desde pago.html, pero en el caso de uso en que se introduce una tarjeta cuya fecha de caducidad ha expirado.

Al principio comienza en pago.html con una petición. Esta pasa al servlet ComienzaPago que recibe la petición y comprobará que los datos están bien. Esto se hará mediante formdatosvisa.jsp, que es donde se introducirán los datos de la tarjeta. A continuación, los datos van al servlet ProcesaPago que, tras comprobar que los datos no son correctos, dará error, llamando a error/muestra error.jsp.

Cuestión número 2:

De los diferentes servlets (recuerde que las páginas jsp también se compilan a un servlet) que se usan en la aplicación, ¿podría indicar cuáles son los encargados de obtener la información sobre el pago con tarjeta cuando se usa pago.html para realizar el pago, y cuáles son los encargados de procesarla? ¿Qué información obtiene y procesa cada uno?

ComienzaPago es el servlet que obtiene la información, ya que es el que utiliza formdatosvisa.jsp. ProcesaPago es el encargado de procesar la información y tanto de comprobar los datos de la tarjeta como de hacer el pago.

Cuestión número 3:

¿Dónde se crea la instancia de la clase pago cuando se accede por pago.html? ¿Y cuándo se accede por testbd.jsp? Respecto a la información que manejan, ¿cómo la comparte entre los distintos servlets? ¿dónde se almacena? ¿dónde se crea ese almacén?

Cuestión número 4:

Enumere las diferencias que existen en la invocación de servlets, a la hora de realizar el pago, cuando se utiliza la página de pruebas extendida testbd.jsp frente a cuando se usa pago.html. ¿Podría indicar por qué funciona correctamente el pago cuando se usa testbd.jsp a pesar de las diferencias observadas?

Mientras que en pago.html se llama al servlet ComienzaPago y le pasa la información de la tarjeta que lleva luego a formdatosvisa.jsp, los datos se guardan en la bean, donde se comprueban los datos con el servlet ProcesaPago; en el caso de testbd.jsp, no se utiliza la el bean, y por lo tanto tiene que solicitar todos los datos para comprobar y procesar la petición.