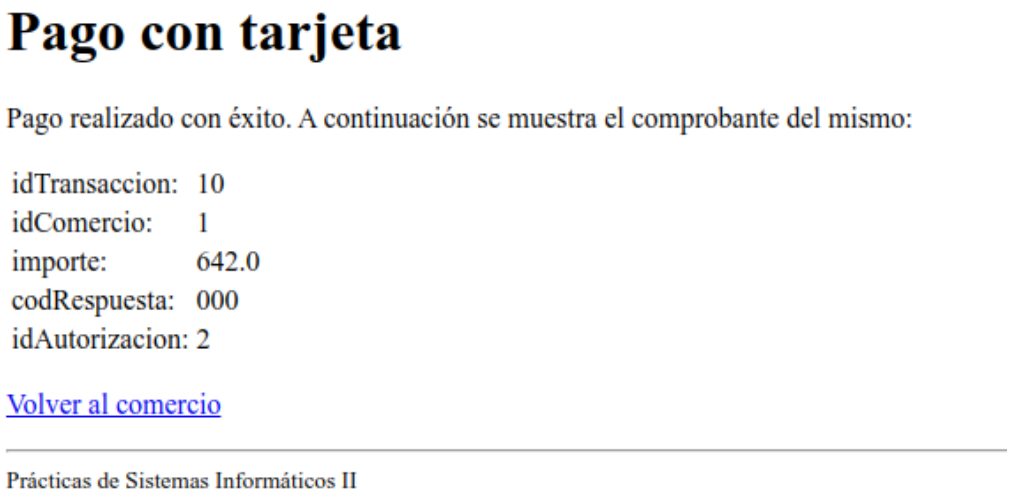
UAM UNIVERSIDAD AUTONOMA DE MADRID		**Escuela Politécnica Superior Ingeniería Informática Práctica de Sistemas Informáticos 1**			
Grupo	**2323**	**Práctica**	**1A**	**Fecha**	27/02/2023
Alumno/a		Hidalgo, Apellido2, Sergio			
Alumno/a		Ibáñez, González, Miguel			

Práctica 1: Arquitectura de JAVA EE

Ejercicio 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.



Pago con tarjeta

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

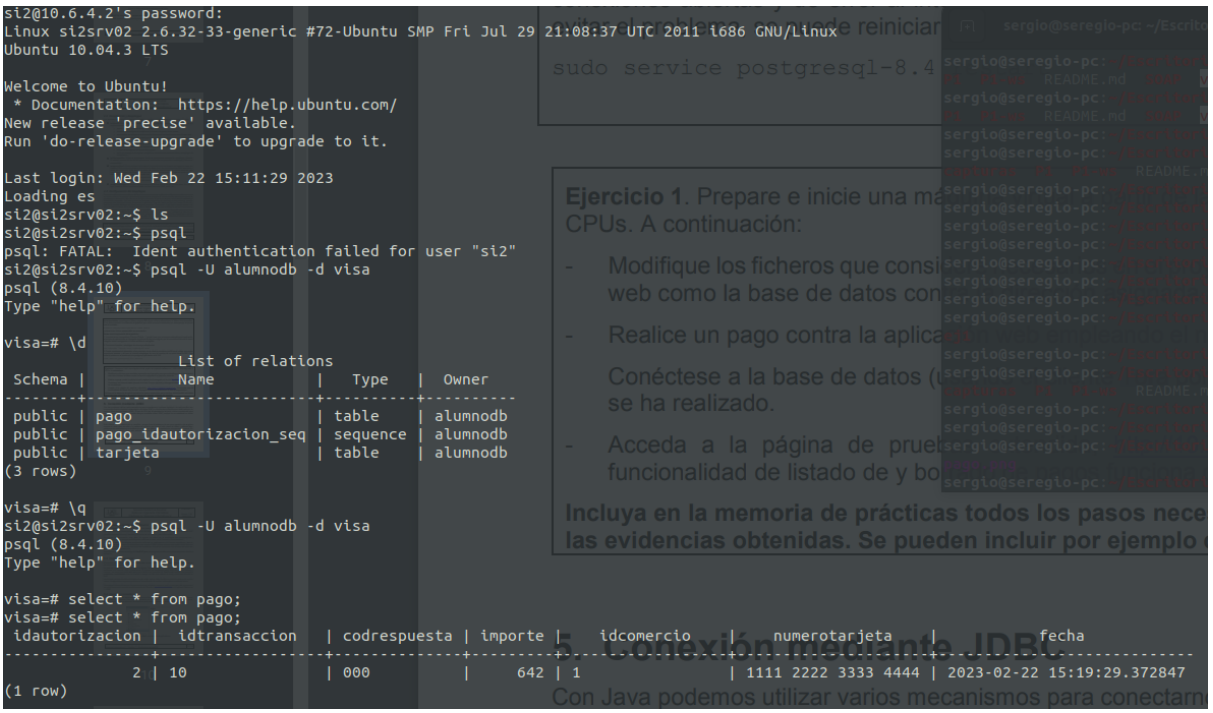
idTransaccion: 10
idComercio: 1
importe: 642.0
codRespuesta: 000
idAutorizacion: 2

Volver al comercio

Prácticas de Sistemas Informáticos II

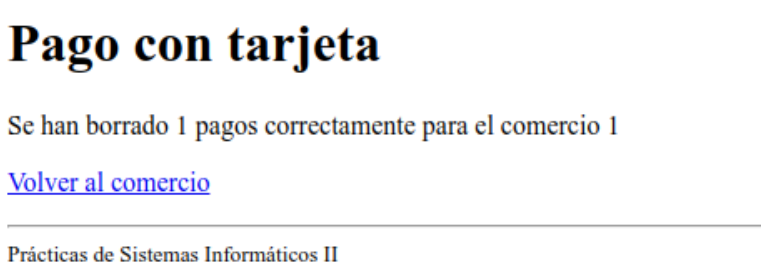
Tras introducir los datos se nos informará de que el pago se ha realizado con éxito y se mostrarán los datos del pago.

Tras conectarse a la máquina virtual a través de una shell remota, nos conectamos a la base de datos y realizamos una consulta (select * from pago;) para comprobar que se ha realizado el pago. Como se puede ver en la captura, solo se muestra una fila, que se corresponde con los datos del pago realizado.



Dentro de testbd.jsp, los mismos datos consultados en la base de datos (a excepción de la fecha y el número de la tarjeta), aparecen al introducir el id de comercio ‘1’ (el mismo que se introdujo al realizar el pago).

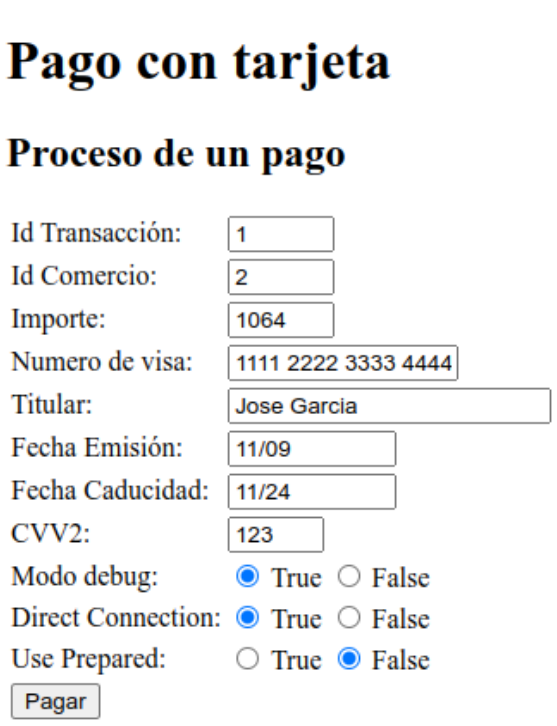
Al introducir este mismo id de comercio, se realizará el borrado de datos y se nos mostrará en un mensaje acerca del estado de la acción realizada (en nuestro caso satisfactorio).

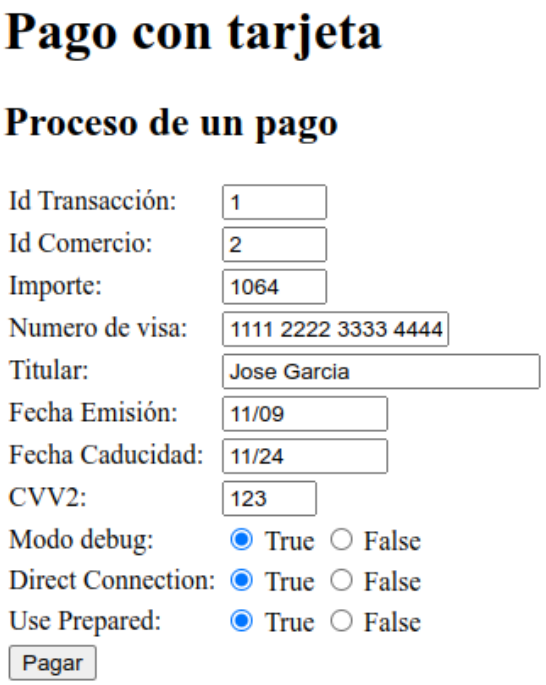


Ejercicio 2:

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 posgresql, 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

Tras realizar la conexión con la página http://10.6.4.2:8080/P1/testbd.jsp se rellenarán los datos del pago para realizar las pruebas. Se testea la conexión directa tanto con consultas preparadas como con no preparadas (se debe tener en cuenta que la creación listado y eliminación del pago, se ha hecho de forma secuencial, es decir, no se ha realizado otro pago existiendo uno previo, por ende no aparecerán ambos pagos en el listado y por ello mismo utilizan los mismos valores).





El resultado para ambos pagos es el mismo, pago satisfactorio.

Pago con tarjeta

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

idTransaccion: 1
idComercio: 2
importe: 1064.0
codRespuesta: 000
idAutorizacion: 5

Volver al comercio

Prácticas de Sistemas Informáticos II

Pago con tarjeta

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

idTransaccion: 1
idComercio: 2
importe: 1064.0
codRespuesta: 000
idAutorizacion: 3

Volver al comercio

Prácticas de Sistemas Informáticos II

Al listar los pagos, se muestra el pago con los valores coincidentes a los introducidos, aunque con distinto id de autorización.

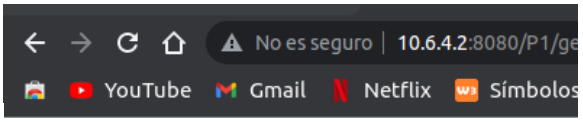
Pago con tarjeta

Lista de pagos del comercio 2

idTransaccion	Importe	codRespuesta	idAutorizacion
1	1064.0	000	5

Volver al comercio

Prácticas de Sistemas Informáticos II



Pago con tarjeta

Lista de pagos del comercio 2

idTransaccion	Importe	codRespuesta	idAutorizacion
1	1064.0	000	3

Volver al comercio

Prácticas de Sistemas Informáticos II

Tras esto se borrará el pago realizado.

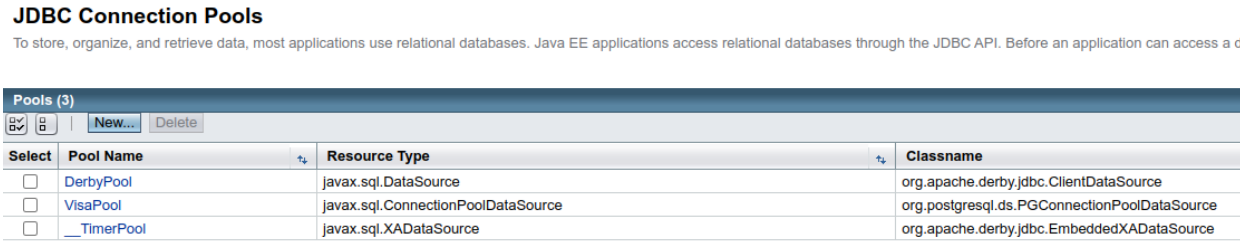
Ejercicio 3:

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.

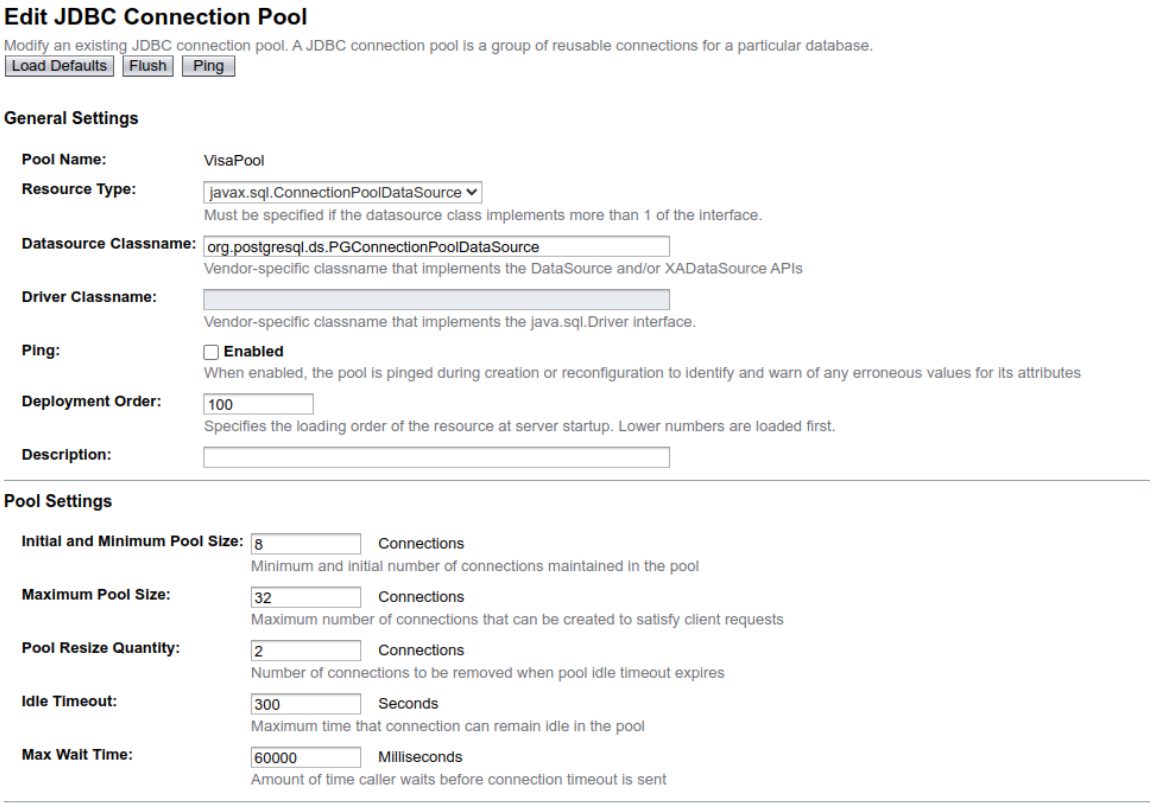
El nombre del recurso para data source se encuentra en la línea 18 de "postgresql.properties"

```
db.datasource=org.postgresql.ds.PGConnectionPoolDataSource
```

Dentro de la Consola de Administración, en el apartado JDBC Connection Pools, dentro de JDBC, en Resources, se puede apreciar la creación de los distintos recursos. El pool que nos interesa es el de data source, fácilmente identificable por la línea anteriormente mencionada, que coincide con su classname. Su nombre es VisaPool.



Al hacer click en el pool name se nos redirigirá a la parte que contiene los valores de dicho recurso. Los ajustes del pool están relacionados con el número de conexiones que se pueden realizar a la base de datos.



Ejercicio 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. Incluya en la memoria de prácticas dichas consultas.

Consulta de si una tarjeta es válida:

En la función compruebaTarjeta en “visaDao.java” linea 133.

```
public boolean compruebaTarjeta(TarjetaBean tarjeta)
```

String con la query preparada en SQL:

```
private static final String SELECT_TARJETA_QRY =
                    "select * from tarjeta " +
                    "where numeroTarjeta=? " +
                    " and titular=? " +
                    " and validaDesde=? " +
                    " and validaHasta=? " +
                    " and codigoVerificacion=? ";
```

Ejecución de la query preparada a través de java:

```
String select  = SELECT_TARJETA_QRY;
              errorLog(select);
              pstmt = con.prepareStatement(select);
              pstmt.setString(1, tarjeta.getNumero());
              pstmt.setString(2, tarjeta.getTitular());
              pstmt.setString(3, tarjeta.getFechaEmision());
              pstmt.setString(4, tarjeta.getFechaCaducidad());
              pstmt.setString(5, tarjeta.getCodigoVerificacion());
              rs = pstmt.executeQuery();
```

Función que devuelve el string en SQL para la query no preparada:

```
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 de la query no preparada a través de java:

```
stmt = con.createStatement();
            qry = getQryCompruebaTarjeta(tarjeta);
            errorLog(qry);
            rs = stmt.executeQuery(qry);
```

Ejecución del pago:

En la función realizaPago en “visaDao.java” linea 106.

```
public synchronized boolean realizaPago(PagoBean pago)
```

String con la query preparada en SQL:

```
private static final String INSERT_PAGOS_QRY =
                    "insert into pago(" +
                    "idTransaccion,importe,idComercio,numeroTarjeta)" +
                    " values (?,?,?,?)";
```

Ejecución de la query preparada a través de java:

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

Función que devuelve el string en SQL para la query no preparada:

```
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;
}
```

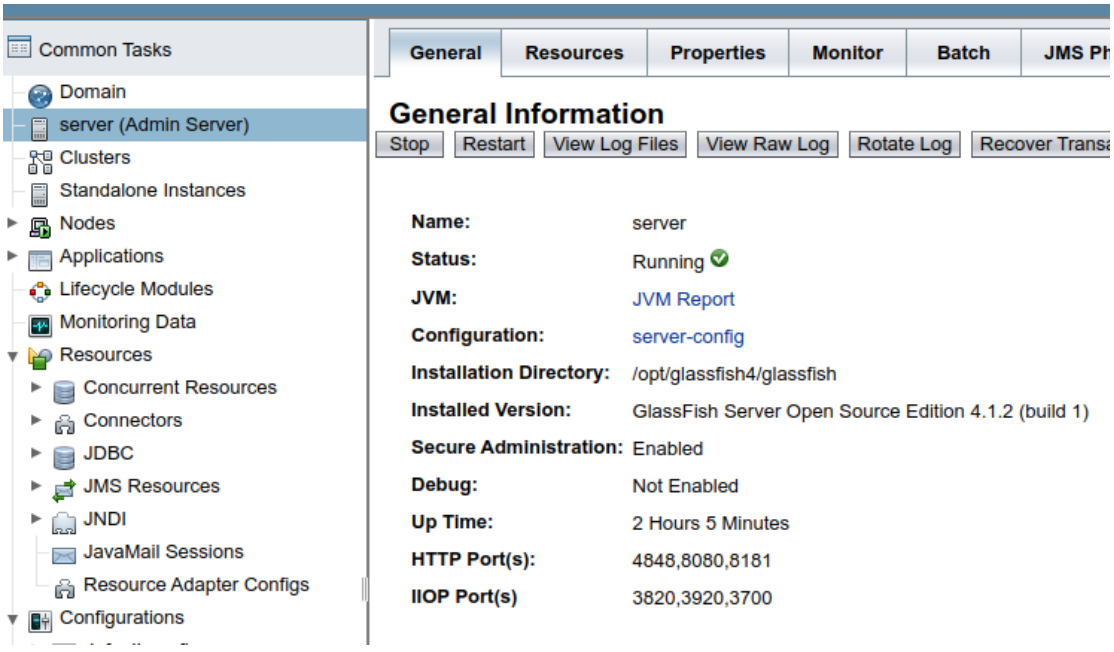
Ejecución de la query no preparada a través de java:

```
stmt = con.createStatement();
            String insert = getQryInsertPago(pago);
            errorLog(insert);
            ret = false;
            if (!stmt.execute(insert)
                    && stmt.getUpdateCount() == 1) {
                ret = true;
```

Ejercicio 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. Incluya en la memoria capturas de pantalla del log del servidor, accediento a él tanto desde el .terminal como del portal web.

En el directorio "/opt/glassfish4/glassfish/domains/domain1/logs/" se encuentra el fichero "server.log", que contendrá toda la información acerca de las actividades realizadas en el servidor.
En la Consola de Administración se puede encontrar dentro del apartado server, haciendo click en "View Log Files".



Tras realizar un pago con el modo debug activado se mostrarán los siguientes mensajes
Se muestran las distintas querys realizadas en la base de datos al realizar el pago.

En el log visto desde terminal (la más antigua primero):

```
[2023-02-23T00:36:28.760-0800] [glassfish 4.1] [SEVERE] [] [] [tid: _ThreadID=29 _ThreadName=Thread-9] [time
Millis: 1677141388760] [levelValue: 1000] [[
  [directConnection=true] select * from tarjeta where numeroTarjeta=?  and titular=?  and validaDesde=?  and
 validaHasta=?  and codigoVerificacion=?]]

[2023-02-23T00:36:28.846-0800] [glassfish 4.1] [SEVERE] [] [] [tid: _ThreadID=29 _ThreadName=Thread-9] [time
Millis: 1677141388846] [levelValue: 1000] [[
  [directConnection=true] insert into pago(idTransaccion,importe,idComercio,numeroTarjeta) values (?,?,?,?)]
]

[2023-02-23T00:36:28.851-0800] [glassfish 4.1] [SEVERE] [] [] [tid: _ThreadID=29 _ThreadName=Thread-9] [time
Millis: 1677141388851] [levelValue: 1000] [[
  [directConnection=true] select idAutorizacion, codRespuesta  from pago  where idTransaccion = ? and idCome
rcio = ?]]
```

En el log visto desde terminal (la más nueva primero):

1557	SEVERE	[directConnection=true] select idAutorizacion, codRespuesta from pago where idTransaccion = ? and ... (details)
1556	SEVERE	[directConnection=true] insert into pago(idTransaccion,importe,idComercio,numeroTarjeta) values (?,?... (details)
1555	SEVERE	[directConnection=true] select * from tarjeta where numeroTarjeta=? and titular=? and validaDesde=... (details)
1554	INFO	visiting unvisited references(details)
1553	INFO	WebModule[null] ServletContext.log():[INFO] Acceso correcto:/procesapago(details)
1552	INFO	WebModule[null] ServletContext.log():[INFO] Acceso correcto:/testbd.jsp(details)

		Feb 23, 2023 00:36:28.851	{levelValue=1000, timeMillis=1677141388851}
		Feb 23, 2023 00:36:28.846	{levelValue=1000, timeMillis=1677141388846}
		Feb 23, 2023 00:36:28.760	{levelValue=1000, timeMillis=1677141388760}
	javax.enterprise.system.tools.deployment.dol	Feb 23, 2023 00:36:27.498	{levelValue=800, timeMillis=1677141387498}
	javax.enterprise.web	Feb 23, 2023 00:36:27.159	{levelValue=800, timeMillis=1677141387159}
	javax.enterprise.web	Feb 23, 2023 00:31:30.671	{levelValue=800, timeMillis=1677141090671}

Ejercicio 6:

Realícense 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. Incluye en la memoria cada fragmento de código donde se han ido añadiendo las modificaciones requeridas. Por último, conteste a la siguiente pregunta:

Líneas 23-25: imports jsw.
Línea 30: etiqueta WebService con el nombre del servicio y el espacio.
Línea 31: cambio de nombre de la clase.
Línea 81: cambio de nombre del constructor.
Líneas 88, 102, 120, 343, 419, 511 y 503: exclude
Líneas 137, 218, 472, 477, 487, 495, 521 y 530: etiquetas WebMethod con el nombre de la operación.
Líneas 138, 219, 478, 496 y 532: etiquetas WebParam
Líneas 521 - 525: override de “isDirectConnection”.
Líneas 530 - 535: override de “setDirectConnection”.
Líneas 219, 223: cambio del retorno de boolean a PagoBean.
Líneas 223, 233, 257, 260, 268, 271, 303 y 310: cambio del retorno, si falso null, si true pago.

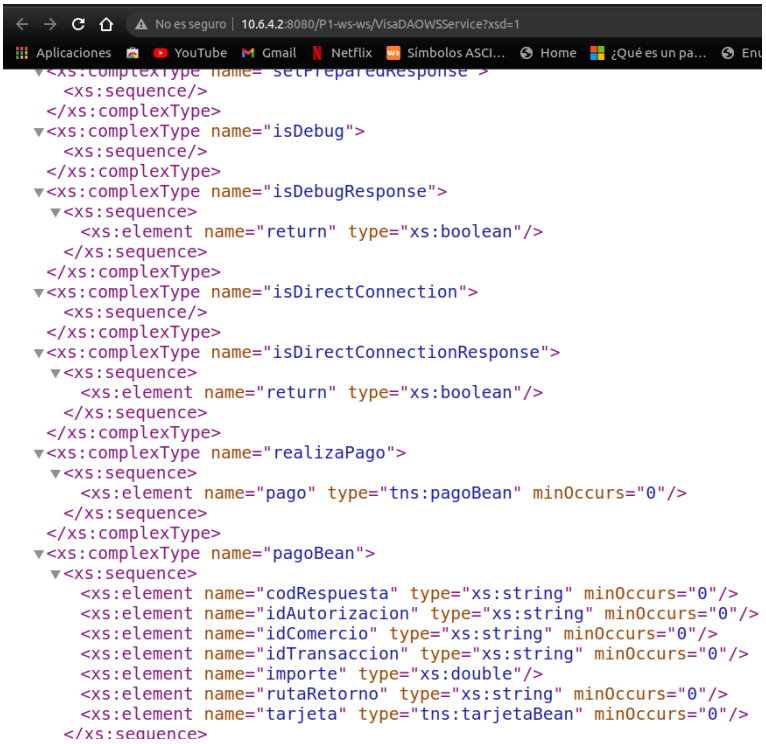
¿Por qué se ha de alterar el parámetro de retorno del método realizaPago() para que devuelva el pago el lugar de un boolean?

Se ha de alterar para pasar la información de vuelta al cliente (pues al separar este método del anterior paquete en el que estaba contenido, el cliente no tendrá acceso a la autorización y el código de respuesta sin el pago modificado).

Ejercicio 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). Comente en la memoria aspectos relevantes del código XML del fichero WSDL y su relación con los métodos Java del objeto del servicio, argumentos recibidos y objetos devueltos 5 . Conteste a las siguientes preguntas:

¿En qué fichero están definidos los tipos de datos intercambiados con el webservice?
En el fichero VisaDAOWSService-wdsl.xml, se puede acceder a él a través de la url: "http://10.6.4.2:8080/P1-ws-ws/VisaDAOWSService?xsd=1"
En la imagen siguiente se muestra el fichero y la definición de PagoBean.



¿Qué tipos de datos predefinidos se usan?
Los tipos de datos predefinidos que se usan son xs:string, xs:double y xs:boolean.

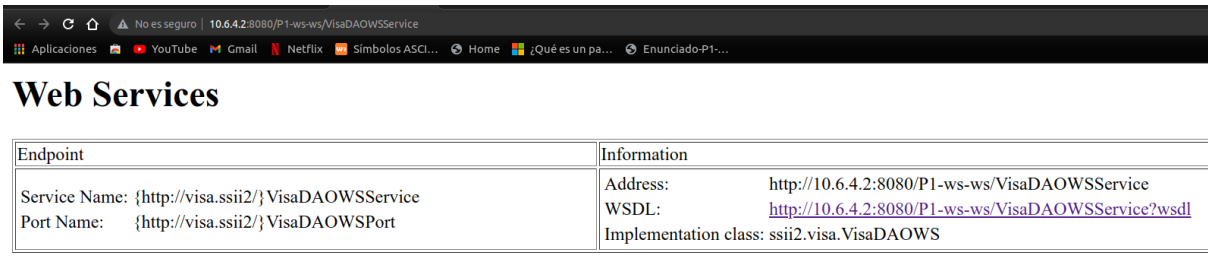
¿Cuáles son los tipos de datos que se definen?
Son tarjetaBean y pagoBean

¿Qué etiqueta está asociada a los métodos invocados en el webservice?
Dentro de "http://10.6.4.2:8080/P1-ws-ws/VisaDAOWSService?wsdl" se asocia "<operation>" a los métodos del webservice.

¿Qué etiqueta describe los mensajes intercambiados en la invocación de los métodos del webservice?
Dentro del mismo fichero, se usa la etiqueta “<message>”

¿En qué etiqueta se especifica el protocolo de comunicación con el webservice?
La etiqueta “<binding>”.

¿En qué etiqueta se especifica la URL a la que se deberá conectar un cliente para acceder al webservice?
Dentro de “<soap:address>” en el parámetro location de la misma (en nuestro caso “http://10.6.4.2:8080/P1-ws-ws/VisaDAOWSService”).



En esta captura se muestra el contenido al acceder a dicha url.

Ejercicio 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. Incluye en la memoria una captura con dichas modificaciones.

Se modifica la creación de VisaDAO por los stubs (de la forma explicada en el enunciado).

```
@Override
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

    TarjetaBean tarjeta = null;
    ValidadorTarjeta val = null;
    PagoBean pago = null;
    VisaDAOWSService service = null;
    VisaDAOWS dao = null;

    tarjeta = creaTarjeta(request);
    val = new ValidadorTarjeta();

    // printAddresses(request,response);
    if (!val.esValida(tarjeta)) {
        request.setAttribute(val.getErrorName(), val.getErrorVisa());
        reenvia(ruta: "/formdatosvisa.jsp", request, response);
        return;
    }

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

Se cambia el control de error de realizaPago.

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

Se coloca el código que utiliza el servicio dentro de un “try” y se recoge la excepción que pueda soltar con un “catch”

```
} catch (Exception e) {
    if (dao.isDebug()) {
        System.err.println("[procesaPago=" + dao.isDirectConnection() + "] " +
                e);
    }
}
```

Ejercicio 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.

Modificación del archivo web.xml para incluir la url del servicio dentro de un parámetro (al igual que se hace con las rutas excluidas).

```
<context-param>
    <param-name>webmaster</param-name>
    <param-value>http://10.6.4.2:8080/P1-ws-ws/VisaDAOWSService</param-value>
</context-param>
```

Uso del parámetro incluido previamente en ProcesaPago.java (dentro de processRequest).

```
BindingProvider bp = (BindingProvider) dao;
bp.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
        getServletContext().getInitParameter("webmaster"));
```

Ejercicio 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. Incluye en la memoria una captura con las adaptaciones realizadas.

getPagos(): Al cambiar el tipo de retorno se modifica en la función la variable retornada, y también se deberá aplicar la función toArray, pero esta vez desde el cliente (haciendo el casting correspondiente). Aparte se deberá colocar las etiquetas webMethod, webParam dentro del fichero VisaDAOWS.java y en el fichero GetPagos, se tendrá que capturar la posible excepción y utilizar el servicio como previamente se ha hecho.

```
 */

@WebMethod(operationName = "getPagos")
public ArrayList<PagoBean> getPagos(@WebParam(name = "idComercio") String idComercio) {
    PreparedStatement pstmt = null;
    Connection pcon = null;
    ResultSet rs = null;
    ArrayList<PagoBean> pagos = null;
    String qry = null;
```

```
*/
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

    VisaDAOWS dao = null;
    try{
       VisaDAOWSService service = new VisaDAOWSService();
       dao = service. getVisaDAOWSPort();
       BindingProvider bp = (BindingProvider) dao;
       bp.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
       getServletContext().getInitParameter("webmaster"));


   } catch (Exception e) {
       System.err.println("[getPagos] " + e);
    }

    /* Se recoge de la petici&oacute;n el par&aacute;metro idComercio*/
    String idComercio = request.getParameter(PARAM_ID_COMERCIO);

    /* Petici&oacute;n de los pagos para el comercio */
    PagoBean[] pagos = (PagoBean[]) ((ArrayList<PagoBean>)
    dao.getPagos(idComercio)).toArray(new PagoBean[dao.getPagos(idComercio).size()]);

    request.setAttribute(ATTR_PAGOS, pagos);
    reenvia(ruta: "/listapagos.jsp", request, response);
    return;
}
```

delPagos(): se modifica tanto la función del servicio como la del archivo .java del cliente, repitiendo el mismo procedimiento.

```
@WebMethod(operationName = "delPagos")
public int delPagos(@WebParam(name = "idComercio") String idComercio)
```

```java
*/
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

    VisaDAOWS dao = null;
    try {
        VisaDAOWSService service = new VisaDAOWSService();
        dao = service.getVisaDAOWSPort();
        BindingProvider bp = (BindingProvider) dao;
        bp.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
                getServletContext().getInitParameter("webmaster"));

    } catch (Exception e) {
        System.err.println("[delPago] " +  e);
        return;
    }

    /* Se recoge de la petici&oacute;n el par&aacute;metro idComercio */
    String idComercio = request.getParameter(PARAM_ID_COMERCIO);

    /* Petici&oacute;n de los pagos para el comercio */
    int ret = dao.delPagos(idComercio);
```

Ejercicio 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.

Se ha creado un scritp de bash para ejecutar la importación al directorio de clases local.

```
sergio@seregio-pc:/opt/glassfish4/glassfish/domains/domain1/applications/j2ee-modules/P1-ws$ bash stubs.bash
analizando WSDL...


Generando código...
```

Dentro del script se encuentra el siguiente comando:
wsimport -d ./build/client/WEB-INF/classes -p ssii2.visa
http:/10.6.4.2:8080/P1-ws-ws/VisaDAOWSService?wsdl
En la carpeta build/client/WEB-INF/classes se encuentran los imports del servicio:

```
sergio@seregio-pc:/opt/glassfish4/glassfish/domains/domain1/applications/j2ee-modules/P1-ws/build/client/WEB
-INF/classes/ssii2/visa$ ls
CompruebaTarjeta.class                IsDirectConnectionResponse.class  SetDebugResponse.class
CompruebaTarjeta.java                 IsDirectConnectionResponse.java   SetDebugResponse.java
CompruebaTarjetaResponse.class        IsPrepared.class                  SetDirectConnection.class
CompruebaTarjetaResponse.java         IsPrepared.java                   SetDirectConnection.java
DelPagos.class                        IsPreparedResponse.class          SetDirectConnectionResponse.class
DelPagos.java                         IsPreparedResponse.java           SetDirectConnectionResponse.java
DelPagosResponse.class                ObjectFactory.class               SetPrepared.class
DelPagosResponse.java                 ObjectFactory.java                SetPrepared.java
GetPagos.class                        package-info.class                SetPreparedResponse.class
GetPagos.java                         package-info.java                 SetPreparedResponse.java
GetPagosResponse.class                PagoBean.class                    TarjetaBean.class
GetPagosResponse.java                 PagoBean.java                     TarjetaBean.java
IsDebug.class                         RealizaPago.class                 VisaDAOWS.class
IsDebug.java                          RealizaPago.java                  VisaDAOWS.java
IsDebugResponse.class                 RealizaPagoResponse.class         VisaDAOWSService.class
IsDebugResponse.java                  RealizaPagoResponse.java          VisaDAOWSService.java
IsDirectConnection.class              SetDebug.class
IsDirectConnection.java               SetDebug.java
```

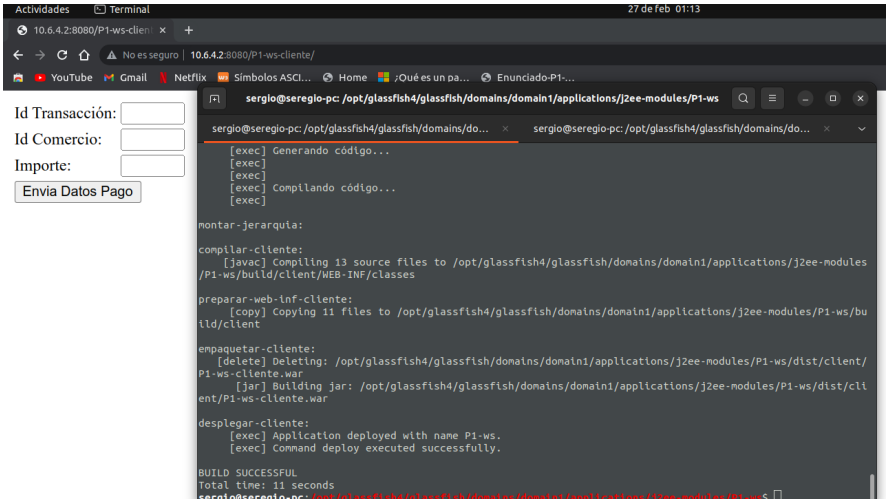
Ejercicio 12.

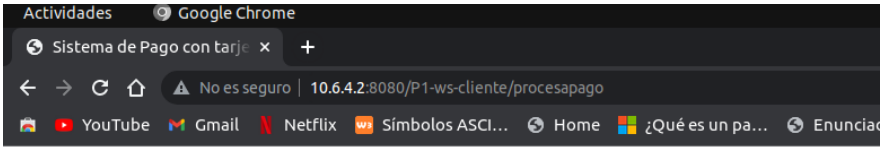
Complete el target generar-stubs definido en build.xml para que invoque a wsimport (utilizar la funcionalidad de ant exec para ejecutar aplicaciones). Téngase en cuenta que: - El raíz del directorio de salida del compilador para la parte cliente ya está definido en build.properties como ${build.client}/WEB-INF/classes - El paquete Java raíz (ssii2) ya está definido como ${paquete} - La URL ya está definida como ${wsdl.url

Después de poner “<exec>” declarando como parámetro interno “executable” el comando wsimport, se rellenan las líneas, según el comando previamente mencionado, primero con -d para indicar el directorio al que se va a importar el paquete del servicio, y más tarde, la flag -p, el paquete (seguido de .visa pues es el único módulo que existe en el servicio) y la url .

```
<target name="generar-stubs" depends="montar-jerarquia" description="Genera los stubs del cliente a parti
  <!-- DONE - Implementar llamada wsimport -->
  <delete file="${build}/${tmpvisaclientjar}" />
  <jar jarfile="${build}/${tmpvisaclientjar}" >
  <fileset dir="${build.client}/WEB-INF/classes" />
  </jar>
  <move file="${build}/${tmpvisaclientjar}" todir="${build.client}/WEB-INF/lib" />
  <exec executable="${wsimport}">
      <arg line=" -d ${build.client}/WEB-INF/classes" />
      <arg line=" -p ${paquete}.visa ${wsdl.url}" />
  </exec>
  </target>
```

Como se puede ver en la siguiente capturas, tras esta modificación, al hacer ant todo, se despliega el cliente correctamente y se pueden realizar pagos.





Pago con tarjeta

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

idTransaccion: 1
idComercio: 1
importe: 1.0
codRespuesta:
idAutorizacion:

Volver al comercio

Prácticas de Sistemas Informáticos II

Ejercicio 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. Incluye evidencias en la memoria de la realización del ejercicio

Cuestiones:

Cuestión 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.

"pago.html" -> servlet "Comienza Pago" (ComienzaPago.java) reenvía a "formdatosvisa.jsp" -> servlet "Procesa Pago" (ProcesaPago.java) -> VisaDAO (VisaDAO.java) comprueba si la tarjeta es válida (no lo es), lanza una excepción y redirige a "error/muestraerror.jsp",

Cuestión 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?

El servlet "Comienza Pago" (ComienzaPago.java) que reenvía a "formatosvista.jsp", que obtendrá información acerca de la tarjeta. Después, el servlet "Procesa Pago" (ProcesaPago.java), mediante la instancia "VisaDao", procesará si hay un error o no, si lo hay se redirigirá a "formatosvista.jsp" con el pertinente error, sino, el usuario será redirigidp a "pagoexito.jsp".

Cuestión 3.

¿Dónde se crea la instancia de la clase pago cuando se accede por pago.html?
La instancia de la clase pago se crea en creaPago, función llamada desde processRequest dentro del servlet "Comienza Pago" (ComienzaPago.java) haciendo uso de PagoBean (clase que representa la información del pago con todos sus atributos).

¿Y cuándo se accede por testbd.jsp?
SI se accede por "testbd.jsp", se creará en el servlet "Procesa Pago" (ProcesaPago.java), al detectar que la sesión devuelve un pago nulo (es decir que no se ha accedido desde pago.html) se creará.

Respecto a la información que manejan, ¿cómo la comparte entre los distintos servlets?
La información se comparte mediante sesiones.

¿dónde se almacena? ¿dónde se crea ese almacén?
La información se almacenará una vez recibida en una base de datos de postgresql, dentro de donde esté alojado el servidor (en nuestro caso en la máquina virtual).

Cuestión 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.

Principalmente, la sesión no ha guardado un pago (al estar la información de la tarjeta y de dicho pago en la misma página, es decir, no necesita pasar la información pues se introduce toda en la misma página) y por ello se salta el uso del servlet “Comienza Pago” (ComienzaPago.java) y los .jsp relacionados con él. Además, se da la opción de elegir el modo debug (se mostrará mayor información sobre los fallos), el tipo de query de java (preparada o no preparada) y la conexión con la base de datos (si va a ser directa o no).

¿Podría indicar por qué funciona correctamente el pago cuando se usa testbd.jsp a pesar de las diferencias observadas?

Como ya se ha explicado anteriormente, al ser el pago obtenido de la sesión nulo, se crea uno nuevo. Como curiosidad acerca de la implementación, si se introducen los datos del pago en “pago.html”, luego en “testbd.jsp” independientemente de los datos de pago que se introduzcan, se detectará el pago de la sesión y se
utilizarán los datos introducidos en “pago.html”,