

 UNIVERSIDAD AUTÓNOMA DE MADRID	Escuela Politécnica Superior Ingeniería Informática Prácticas de Sistemas Informáticos 2				
Grupo	2311	Práctica	1B	Fecha	28/03/2022
Alumno/a	Fraile, Iglesias, Javier				
Alumno/a	Fernández, París, Iván				

Práctica 1B: Arquitectura de JAVA EE (Segunda parte)

Ejercicio número 1:

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

- Hacer que la clase implemente la interfaz local y convertirla en un EJB stateless mediante la anotación Stateless
- Eliminar el constructor por defecto de la clase
- Ajustar el método getPagos() a la interfaz definida en VisaDAOLocal

A continuación, se mostrarán imágenes que reflejan que los cambios comentados han sido aplicados.

```
@Stateless (mappedName = "VisaDaoBean")  
  
public class VisaDAOBean extends DBTester implements VisaDAOLocal{
```

De esta manera, se implementa la interfaz local con los métodos que va a implementar nuestro servicio.

Los cambios en el método de “getPagos” (que ya no devuelve un arrayList) son la cabecera:

```
public PagoBean[] getPagos (String idComercio) {  
  
    PagoBean [] ret = null;  
  
    ArrayList<PagoBean> pagos = new ArrayList<>();
```

Y a continuación la conversión del array:

```
//Se convierte el arrayList a array de Pago Bean  
  
ret = new PagoBean[pagos.size()];  
  
ret = pagos.toArray(ret);
```

Ejercicio número 2:

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

- En la sección de preproceso, añadir las siguientes importaciones de clases que se van a utilizar:

```
import javax.ejb.EJB;
import ssii2.visa.VisaDAOLocal;
```

- Añadir como atributo de la clase el objeto proxy que permite acceder al EJB local, con su correspondiente anotación que lo declara como tal:

```
@EJB (name = "VisaDAOBean", beanInterface=VisaDAOLocal.class)
VisaDAOLocal dao = null;
```

Se ha eliminado toda la información relevante a la instancia de “VisaDAOWS” así como “BindingProvider”, porque ahora tan solo necesitamos la instancia a la interfaz pública EJB que contienen todos los métodos que se ejecutan en local.

Finalmente, se ha modificado dentro del servlet de “GetPagos”, la llamada al método de “getPagos” de “VisaDaoBean”, ya que ahora devuelve tan solo un array:

```
PagoBean[] pagos = dao.getPagos(idComercio);
```

Ejercicio número 3:

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

- Editar el archivo build.properties para que la propiedad as.host contenga la dirección IP del servidor de aplicaciones. Indica el valor y por qué es ese valor.

```
as.user=admin
as.host=10.4.6.2
as.port=4848
```

El valor del as.host, va a ser del servidor de aplicaciones donde vamos a desplegar la aplicación, en nuestro caso, el 10.4.6.2 porque como comenta el enunciado, es el que usaremos.

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

```
db.port=5432
db.host=10.4.6.1
```

Esto en cuanto al host, ya que la base de datos reside en el 10.4.6.1, como indica el enunciado.

```
db.client.host=10.4.6.2
db.client.port=4848
```

El host del cliente es el de nuestro servidor de aplicaciones a donde nos tenemos que conectar, en nuestro caso la 10.4.6.2.

Desplegar la aplicación de empresa

Una vez desplegamos la aplicación, accedemos a la ip 10.4.6.2, que es donde establecimos el servidor de aplicaciones y obtenemos lo esperado:

The screenshot shows the GlassFish administration interface. On the left, a sidebar lists various server components like Domain, Clusters, Nodes, and Applications. Under Applications, 'P1-ejb' is selected. The main panel displays deployment settings for 'P1-ejb'. It includes sections for Implicit CDI (Enabled), Java Web Start (Enabled), Location (set to \${com.sun.aas.instanceRootURL}/applications/P1-ejb/), Deployment Order (set to 100), Libraries, and Description. Below this, a table titled 'Modules and Components (9)' lists the deployed modules and their components. The table has columns for Module Name, Engines, Component Name, Type, and Action (with a 'Launch' link). The listed components include DelPagos, ProcesaPago, GetPagos, ComienzaPago, and VisaDAOBean.

Module Name	Engines	Component Name	Type	Action
P1-ejb-cliente.war	[web]	-----	-----	Launch
P1-ejb-cliente.war		default	Servlet	
P1-ejb-cliente.war		jsp	Servlet	
P1-ejb-cliente.war		DelPagos	Servlet	
P1-ejb-cliente.war		ProcesaPago	Servlet	
P1-ejb-cliente.war		GetPagos	Servlet	
P1-ejb-cliente.war		ComienzaPago	Servlet	
P1-ejb.jar	[ejb, weld]	-----	-----	
P1-ejb.jar		VisaDAOBean	StatelessSessionBean	

Ejercicio número 4:

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

Si la base de datos no se ha generado previamente, será necesario crearla usando ant regenerar-bd

1. Mediante pago.html
 - a. Accedemos a la URL "<http://10.4.6.2:8080/P1-ejb-cliente/>" y obtenemos el template base, lo rellenamos y enviamos datos de pago:

The screenshot shows a web page with a form for a payment transaction. The form fields are: Id Transacción: 1, Id Comercio: 1, Importe: 16, and a button labeled 'Envia Datos Pago'.

- b. Rellenamos los datos del titular que va a realizar la transacción como se muestra en la imagen:

No es seguro | 10.4.6.2:8080/P1-ejb-cliente/comienzapago

Gmail GitHub GitLab UAM YouTube Moodle UAM W3 IA Maps Twitter

Pago con tarjeta

Numero de visa:

Titular:

Fecha Emisión:

Fecha Caducidad:

CVV2:

Id Transacción: 1
 Id Comercio: 1
 Importe: 16.0

Prácticas de Sistemas Informáticos II

- c. Pulsamos el botón de pagar y observamos que se ha realizado correctamente la transacción.

No es seguro | 10.4.6.2:8080/P1-ejb-cliente/procesapago

Gmail GitHub GitLab UAM YouTube Moodle UAM W3 IA Maps Twitter

Pago con tarjeta

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

idTransaccion: 1
 idComercio: 1
 importe: 16.0
 codRespuesta: 000
 idAutorizacion: 1

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

- d. Comprobamos en la base de datos que la transacción se realiza con éxito, en nuestro caso condbeaver:

The screenshot shows the pgAdmin III application interface. On the left, there's a tree view of database objects under 'Bases de Datos'. The 'pago' table is selected. On the right, the table structure is shown with columns: idautorizacion, idtransaccion, codrespuesta, importe, idcomercio, numerotarjeta, and fecha. A single row is displayed with values: 1, 000, 16, 1, 1111 2222 3333 4444, and 2022-03-07 01:50:35.250.

2. Mediante testdb.jsp

- a. Accedemos a la URL "<http://10.4.6.2:8080/P1-ejb-cliente/testbd.jsp>", y rellenamos el formulario como se muestra a continuación:

No es seguro | 10.4.6.2:8080/p1-ejb-cliente/testbd.jsp

Gmail GitHub GitLab UAM YouTube Moodle UAM W3 IA Maps Twitter

Pago con tarjeta

Proceso de un pago

Id Transacción:	2
Id Comercio:	2
Importe:	20
Número de visa:	0694 4853 5696 209
Titular:	Rosita Torres Coll
Fecha Emisión:	08/09
Fecha Caducidad:	03/23
CVV2:	547
Modo debug:	<input checked="" type="radio"/> True <input type="radio"/> False
Direct Connection:	<input type="radio"/> True <input checked="" type="radio"/> False
Use Prepared:	<input type="radio"/> True <input checked="" type="radio"/> False

Consulta de pagos

Id Comercio:

Borrado de pagos

Id Comercio:

Prácticas de Sistemas Informáticos II

- b. Una vez rellenamos los datos, podemos observar que la transacción se ha realizado correctamente:

No es seguro | 10.4.6.2:8080/p1-ejb-cliente/procesapago

Gmail GitHub GitLab UAM YouTube Moodle UAM W3 IA Maps Twitter

Pago con tarjeta

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

idTransaccion:	2
idComercio:	2
importe:	20.0
codRespuesta:	000
idAutorizacion:	2

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

- c. Y en la base de datos observamos que también se ha añadido:

Navegador de Bases de Datos X Proyectos

Ingrese parte del nombre de un objeto aquí

Propiedades Datos Diagrama ER

postres Bases de Datos visa public tarjeta pago

postres Bases de Datos visa public Tablas pago

postres - 10.4.6.1:5432

Esquemas

Tables pago

Postgres

Diagrama ER

Grilla

Entera SQL expression to filter results (use Ctrl+Space)

pago

idautorizacion	idtransaccion	codrespuesta	importe	idcomercio	numerotarjeta	fecha
1	2	000	20.0	2	1111 2222 3333 4444	2022-03-07 01:50:35.250
2	2	000	20.0	2	0694 4853 5696 2092	2022-03-07 03:33:48.708

- d. Ahora vamos a probar a listar los registros del id comercio = 2, que es el que hemos añadido:

No es seguro | 10.4.6.2:8080/p1-ejb-cliente/getpagos

Gmail GitHub GitLab UAM YouTube Moodle UAM W3 IA Maps Twitter

Pago con tarjeta

Lista de pagos del comercio 2

idTransaccion	Importe	codRespuesta	idAutorizacion
2	20.0	000	2

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

- e. Finalmente probamos a eliminar el registro del comercio = 2:



f. Comprobamos la base de datos desde eldbeaver y vemos si se ha borrado:

Ejercicio número 5:

Realizar los cambios indicados en P1-ejb-servidor-remoto y preparar los PCs con el esquema de máquinas virtuales indicado. Compilar, empaquetar y desplegar de nuevo la aplicación P1-ejb como servidor de EJB remotos de forma similar a la realizada en el Ejercicio 3 con la Figura 2 como entorno de despliegue. Esta aplicación tendrá que desplegarse en la máquina virtual del PC2.

Se recomienda replegar la aplicación anterior (EJB local) antes de desplegar ésta.

En primer lugar hemos copiado la clase de “VisaDAOLocal” en un nuevo fichero “VisaDAORemote”, y cambiar el tipo de implementación de la interfaz de @Local a @Remote:

```
import javax.ejb.Remote;

@Remote

public interface VisaDAORemote {
```

A continuación, hacemos que “VisaDAOBean” implementa también la nueva interfaz remota:

```
@Stateless(mappedName = "VisaDAOBean")

public class VisaDAOBean extends DBTester implements VisaDAOLocal,
VisaDAORemote
```

Además, hemos hecho que las clases “PagoBean” y “TarjetaBean” implementen la interfaz serializable de la siguiente manera:

```
import java.io.Serializable;

public class TarjetaBean implements Serializable{
```

```
import java.io.Serializable;

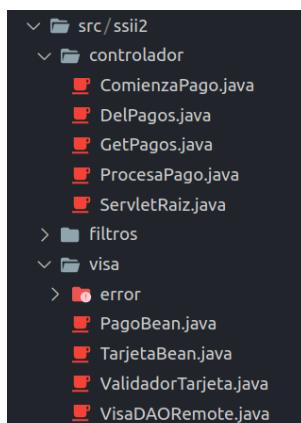
public class PagoBean implements Serializable{
```

Finalmente, hemos vuelto a compilar, empaquetar y desplegar la aplicación como habíamos hecho en el primer apartado y observamos que se ha desplegado correctamente:

Ejercicio número 6:

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

Se ha eliminado en la nueva copia del cliente todo lo referente al servidor que había y el esquema de ficheros en árbol del cliente-remoto es la siguiente:



Como se puede observar, ya no existe ni las carpetas ni cliente ni servidor, ya que solo queda el cliente.

En la carpeta Visa, hemos copiado las clases que había en el servidor de “TarjetaBean.java” y “PagoBean.java”, además de hacerlas serializables de la misma manera que se hizo en apartados anteriores.

Finalmente, se ha copiado el archivo “VisaDAORemote.java”, así podemos hacer referencia a ese objeto de manera remota.

Se ha añadido en los controladores de “ProcesaPago”, “DelPagos” y “GetPagos” la referencia al objeto remoto de la siguiente manera:

```
@EJB(name = "VisaDAOBean", beanInterface=VisaDAORemote.class)

private VisaDAORemote dao = null;
```

Por otro lado, se ha creado el fichero glassfish-web, el cual hace referencia al objeto remoto (implementando con el protocolo de comunicación entre aplicaciones por internet mediante CORBA), que se encuentra en el servidor de aplicaciones desplegado en el host 10.4.6.2 en el anterior apartado.

Finalmente, la base de datos se encuentra en el host 10.4.6.1, y el client host, es el host donde está localizado el servidor de aplicaciones que es el que hará consultas a esta base de datos, en nuestro caso 10.4.6.2.

A continuación, compilamos, empaquetamos y subimos la aplicación, y la lanzamos en el host del cliente con la siguiente URL: "<http://10.4.6.1:8080/P1-ejb-cliente-remoto/>" y ahora realizaremos el pago:

No es seguro | 10.4.6.1:8080/P1-ejb-cliente-remoto/

Gmail GitHub GitLab UAM YouTube Moodle UAM

Id Transacción:	2
Id Comercio:	2
Importe:	14
<input type="button" value="Envia Datos Pago"/>	

No es seguro | 10.4.6.1:8080/P1-ejb-cliente-remoto/comenzapago

Gmail GitHub GitLab UAM YouTube Moodle UAM W3 IA Maps Twitter

Pago con tarjeta

Numero de visa:

Titular:

Fecha Emisión:

Fecha Caducidad:

CVV2:

Id Transacción: 2

Id Comercio: 2

Importe: 14.0

Prácticas de Sistemas Informáticos II

Pago con tarjeta

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

idTransaccion: 2
idComercio: 2
importe: 14.0
codRespuesta:
idAutorizacion:

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

Como vemos, se ha realizado el pago correctamente a la conexión remota, ahora comprobaremos en el gestor de la base de datos, en el 10.4.6.1 desdedbeaver:

Ejercicio número 7:

Modificar la aplicación VISA para soportar el campo saldo:

Archivo TarjetaBean.java:

· Añadir el atributo saldo y sus métodos de acceso:

En primer lugar, vamos a crear el atributo privado de clase del nuevo campo con el que vamos a jugar en este apartado, que es el saldo, así como sus métodos getter y setter para poder ser manipulado:

```
private double saldo;
```

```
public void setSaldo(Double saldo) {
    this.saldo = saldo;
}

public Double getSaldo() {
    return saldo;
}
```

Archivo VisaDAOBean.java:

· Importar la definición de la excepción EJBException que debe lanzar el servlet para indicar que se

debe realizar un rollback:

```
import javax.ejb.EJBException;
```

· Declarar un prepared statement para recuperar el saldo de una tarjeta de la base de datos.

Para la consulta, recogemos el saldo de una tarjeta específica, a partir de su número, que es un atributo único en la base de datos.

```
private static final String GET_SALDO_TARJETA_QRY =  
  
        "select saldo from tarjeta " +  
  
        " where numeroTarjeta=?";
```

· Declarar un prepared statement para actualizar el nuevo saldo calculado en la base de datos.

Para la consulta, actualizamos el campo de saldo de la tarjeta , a partir de su número.

```
private static final String UPDATE_SALDO_TARJETA_QRY =  
  
        "update tarjeta set saldo=? " +  
  
        " where numeroTarjeta=?";
```

· Modificar el método realizaPago con las siguientes acciones:

En primer lugar, al establecer la conexión, se va a ejecutar la primera consulta a la base de datos para obtener el saldo del cliente a partir del número de la tarjeta asociado.

```
try {  
  
    con = getConnection();  
  
    String saldoQuery = GET_SALDO_TARJETA_QRY;  
  
    errorLog(saldoQuery);  
  
  
    pstmt = con.prepareStatement(saldoQuery);  
  
    pstmt.setString(1, pago.getTarjeta().getNumero());  
  
    rs = pstmt.executeQuery();
```

A continuación, desplazamos el cursor al primer registro de la consulta respuesta obtenida, y cogemos el campo del saldo.

Si el saldo es mayor o igual que el importe que ha solicitado el cliente, continuamos con la siguiente consulta que es la de actualización del saldo del cliente

Con lo cual el nuevo saldo del cliente será el resultado de restar el saldo actual al importe solicitado

Ejecutamos dicha consulta y en el caso de que no nos haya devuelto un registro, vamos a establecer la flag de ret = false para lanzar la excepción posteriormente

También, en el caso de que el saldo que solicite el cliente sea mayor, vamos a devolver un pago nulo para mostrar un error.

```
if (rs.next()) {  
  
    double saldo = rs.getDouble("saldo");
```

```

    if (saldo >= pago.getImporte()) {

        String saldoInsert = UPDATE_SALDO_TARJETA_QRY;
        errorLog(saldoInsert);

        pstmt = con.prepareStatement(saldoInsert);
        pstmt.setDouble(1, saldo - pago.getImporte());
        pstmt.setString(2, pago.getTarjeta().getNumero());

        ret = false;

        if (!pstmt.execute()
            && pstmt.getUpdateCount() == 1) {
            ret = true;
        }
    } else {
        pago.setIdAutorizacion(null);
        return null;
    }
} else {
    ret = false;
}

```

Como habíamos comentado, en el caso de que el retorno sea falso, lo que quiere decir que la base de datos no se ha actualizado correctamente, vamos a lanzar una EJBException que será capturada en el “ProcesaPago”, para que automáticamente la transacción haga un rollback y establezca el estado anterior al de comenzar la transacción.

```

if (ret == false) {
    throw new EJBException();
}

```

Comentar, que esta comprobación no solo se hace en este punto, si no que también en la segunda transacción (la de crear la entrada del pago en la base de datos posterior), ya que puede haber id de transacción y comercio repetido, y en dicho caso también se deshace la transacción.

- **Modificar el servlet ProcesaPago para que capture la posible interrupción EJBException lanzada por realizaPago, y, en caso de que se haya lanzado, devuelva la página de error mediante el método enviaError (recordar antes de retornar que se debe invalidar la sesión, si es que existe).**

Finalmente, la clase de “procesaPago”, se envuelve en un try-catch que capture la excepción del pago, que hará rollback.

Si hay una sesión activa se cierra, y si el pago es nulo (porque introduzco un importe mayor al saldo) entonces le envío el pago invalido

```
try{  
  
    PagoBean pagoModificado = dao.realizaPago(pago);  
  
    if (pagoModificado == null) {  
  
        enviaError(new Exception("Pago incorrecto"), request,  
                  response);  
  
        return;  
    }  
  
}catch(EJBException e){  
  
    if (sesion != null){  
  
        sesion.invalidate();  
  
    }  
  
    enviaError(e, request, response);  
}
```

Ejercicio número 8:

Desplegar y probar la nueva aplicación creada.

- Probar a realizar pagos correctos. Comprobar que disminuye el saldo de las tarjetas sobre las que realice operaciones. Añadir a la memoria las evidencias obtenidas.

Nos metemos en el cliente a través del host 10.4.6.2, y la base de datos reside en el host 10.4.6.1 de acuerdo a la figura 2 del enunciado.

Introducimos un importe de 25, y todos los los clientes parten de un importe de 1000 en la base de datos:



Numero de visa: 7772 8952 5915 5042
Titular: John Dominguez Lopez
Fecha Emisión: 01/10
Fecha Caducidad: 10/23
CVV2: 317
Pagar

Id Transacción: 1
Id Comercio: 2
Importe: 25.0

Observamos el correcto resultado tanto en la página, como la actualización en la base de datos mediante una consulta:

The screenshot shows a browser window with the URL 10.4.6.2:8080/P1-ejb-cliente/procesopago. The page title is "Pago con tarjeta". The content says "Pago realizado con éxito. A continuación se muestra el comprobante del mismo:". Below this, there is a table of transaction details:

idTransaccion:	1
idComercio:	2
importe:	25.0
codRespuesta:	000
idAutorizacion:	1

At the bottom, there is a link "Volver al comercio".

Pago con tarjeta

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

```
idTransaccion: 1  
idComercio: 2  
importe: 25.0  
codRespuesta: 000  
idAutorizacion: 1
```

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

The screenshot shows a MySQL database interface with the connection string "visa - 10.4.6.1:5432". A query is being run in the SQL editor:

```
select * from tarjeta  
where numeroTarjeta='7772 8952 5915 5042'
```

The results are displayed in a grid:

numeroTarjeta	titular	validadesde	validahasta	codigoverificación	saldo
7772 8952 5915 5042	John Dominguez Lopez	01/10	10/23	317	975

- Realice una operación con identificador de transacción y de comercio duplicados. Compruebe que el saldo de la tarjeta especificada en el pago no ha variado.

En este caso elegimos los mismos id de transacción y de comercio para otro cliente:

The left side shows a payment form with the following values:

Id Transacción:	1
Id Comercio:	2
Importe:	25

There is a button "Envia Datos Pago".

The right side shows the payment confirmation page with the same transaction details. Below the form, the transaction details are listed again:

Numero de visa:	1530 6462 9686 4119
Titular:	Alberto Mas Reyes
Fecha Emisión:	05/09
Fecha Caducidad:	09/23
CVV2:	105

A "Pagar" button is present.

Below the form, the transaction details are listed again:

Id Transacción:	1
Id Comercio:	2
Importe:	25.0

At the bottom, it says "Prácticas de Sistemas Informáticos II".

Comprobamos y no nos manda feedback de que ha sido exitosa la transacción, con lo cual comprobamos la base de datos y observamos que efectivamente se ejecutó un rollback en la operación (Ya que primeramente se actualizó el saldo porque la primera parte de "realizaPago" la hizo bien, pero la segunda parte que crea el registro el pago la hizo mal):

Pago con tarjeta

Prácticas de Sistemas Informáticos II

idautorizacion	idtransaccion	codresuesta	importe	idcomercio	numerotarjeta	fecha
1	1	000	25	2	7772 8952 5915 5042	2022-03-11 10:38:37.165

Y comprobamos en concreto ese cliente para ver que no se haya reducido el saldo en la primera parte de la operación:

numerotarjeta	titular	validadesde	validahasta	codigoverificacion	saldo
1530 6462 9686 4119	Alberto Mas Reyes	05/09	09/23	105	1.000

El saldo volvió a su estado original.

Ejercicio número 9:

En la máquina virtual donde se encuentra el servidor de aplicaciones (10.X.Y.Z2), declare manualmente la factoría de conexiones empleando la consola de administración, tal y como se adjunta en la Figura 4.

New JMS Connection Factory

The creation of a new Java Message Service (JMS) connection factory also creates a connector connection pool for the factory and a connector resource.

General Settings

- JNDI Name:
- Resource Type:
- Description: conexión para la cola de pagos
- Status: Enabled

Pool Settings

- Initial and Minimum Pool Size: Connections
- Maximum Pool Size: Connections
- Pool Resize Quantity: Connections
- Idle Timeout: Seconds
- Max Wait Time: Milliseconds
- On Any Failure: Close All Connections

Ejercicio número 10:

En la máquina virtual donde se encuentra el servidor de aplicaciones (10.X.Y.Z2), declare manualmente la conexión empleando la consola de administración, tal y como se adjunta en la Figura 5.

The screenshot shows the GlassFish administration interface. On the left, there's a tree view of server configurations. The main panel is titled 'Edit JMS Destination Resource'. It contains fields for JNDI Name (set to jms/VisaPagosQueue), Physical Destination Name (VisaPagosQueue), Resource Type (set to javax.jms.Queue), Deployment Order (100), Description (cola de pagos), and Status (checkbox checked for Enabled). Below these fields is a table titled 'Additional Properties (0)' with two buttons: 'Add Property' and 'Delete Properties'. The table itself is empty, showing 'No items found.' At the bottom right of the dialog are 'Save' and 'Cancel' buttons.

Ejercicio número 11:

- Modifique el fichero sun-ejb-jar.xml para que el MDB conecte adecuadamente a su connection factory

```
<mdb-connection-factory>

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

</mdb-connection-factory>
```

- Incluya en la clase VisaCancelacionJMSBean:

- Consulta SQL necesaria para obtener el código de respuesta del pago cuyo idAutorizacion coincide con lo recibido por el mensaje

```
private static final String GET_CODRESPUESTA_QRY =
    "select codrespuesta from pago where " +
    " idautorizacion=?";
```

- Consulta SQL necesaria para actualizar el código de respuesta a valor 999, de aquella autorización existente en la tabla de pagos cuyo idAutorizacion coincide con lo recibido por el mensaje

```
private static final String UPDATE_CANCELAR_QRY =
    "update pago set codrespuesta=999 where " +
    " idautorizacion=?";
```

- Consulta SQL necesaria para rectificar el saldo de la tarjeta que realizó el pago

Para esta consulta, primero se usa el saldo que ya había más el saldo que se obtiene del pago con ese id autorización, por otro lado para saber a qué tarjeta aplicarlo, también se añade la tarjeta que aparece en dicho pago con ese id de autorización.

```
private static final String UPDATE_SALDO_QRY =  
  
        "update tarjeta set saldo = saldo  + " +  
        " (select importe from pago where idautorizacion=?) " +  
        " where numerotarjeta = " +  
        " (select numerotarjeta from pago where idautorizacion=?) ";
```

- Método onMessage() que obtenga el idAutorización de la cola de mensajes, compruebe si el pago con dicho idAutorizacon tiene código de respuesta 000 y en ese caso actualice el código de respuesta y rectifique el saldo de la tarjeta. Para ello tome de ejemplo el código SQL de ejercicios anteriores, de modo que se use un prepared statement que haga bind del idAutorizacion para cada mensaje recibido.

En la primera parte de la consulta, se añaden los atributos necesarios para hacer la conexión a la base de datos y las sentencias preparadas, y se obtiene el id de autorización haciendo un parseo de lo que llega en el mensaje, ya que se corresponde a este id:

```
public void onMessage(Message inMessage) {  
  
    TextMessage msg = null;  
    Connection con = null;  
    PreparedStatement pstmt = null;  
    boolean ret = false;  
    ResultSet rs = null;  
  
    try {  
  
        if (inMessage instanceof TextMessage) {  
  
            msg = (TextMessage) inMessage;  
            logger.info("MESSAGE BEAN: Message received: " +  
                    msg.getText());  
        } else {  
  
            logger.warning(  
                    "Message of wrong type: "  
                    + inMessage.getClass().getName());  
        }  
        int idAutorizacion = Integer.parseInt(msg.getText());
```

A continuación, se obtiene el código de respuesta correspondiente a dicho id de autorización

pueden pasar varias cosas:

1. Que la consulta no haya recibido respuesta, lo que significa que no hay un pago con ese id de autorización, entonces se devuelve error y se comprueba con rs.next().
2. Que el código de respuesta de la consulta no sea 000, lo que indica que sería 999 y el pago ya está devuelto entonces tampoco se entra.

```
con = getConnection();

        //Se prepara la query para obtener el codigo de respuesta
        String codResQuery = GET_CODRESPUESTA_QRY;
        logger.info(codResQuery);

        pstmt = con.prepareStatement(codResQuery);
        pstmt.setInt(1, idAutorizacion);
        rs = pstmt.executeQuery();

        if (rs.next()) {
            int codigoRes = rs.getInt("codresuesta");
            if (codigoRes == 000) {
                .
                .
                .
            } else {
                logger.warning(
                    "Codigo Respues != 000: "
                    + inMessage.getClass().getName());
                return;
            }
        } else {
            logger.warning(
                "IdAutorizacion no encontrado: "
                + inMessage.getClass().getName());
            return;
        }
    }
```

En caso de que el código de respuesta sea 000 entramos en el if y ejecutamos la actualización del código al 999.

```

String updCodResQuery = UPDATE_CANCELAR_QRY;

logger.info(updCodResQuery);

//El nuevo código es 999

pstmt = con.prepareStatement(updCodResQuery);

pstmt.setInt(1, idAutorizacion);

ret = false;

if (!pstmt.execute()

&& pstmt.getUpdateCount() == 1) {

ret = true;

}

```

Finalmente, si el retorno = true, lo que significa que la actualización ha sido exitosa ya solo quedaría actualizar el saldo del cliente involucrado, para ello se hace de manera similar con la consulta de actualizar saldo:

```

if (ret == true) {

String updSaldoQuery = UPDATE_SALDO_QRY;

logger.info(updSaldoQuery);

pstmt = con.prepareStatement(updSaldoQuery);

pstmt.setInt(1, idAutorizacion);

pstmt.setInt(2, idAutorizacion);

ret = false;

if (!pstmt.execute()

&& pstmt.getUpdateCount() == 1) {

ret = true;

}
}

```

Ya solo queda la ultima comprobacion de si el saldo no ha sido actualizado:

```

if (ret == false){

logger.warning(

"Saldo no ha sido actualizado: "

+ inMessage.getClass().getName());

return;
}

```

```
}
```

Ejercicio número 12:

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

En el fichero “VisaQueueMensajeProducer”, se ha añadido con @ antes del método de “browseMessages” la inicialización estática:

```
@Resource(mappedName = "jms/VisaConnectionFactory")  
private static ConnectionFactory connectionFactory;  
  
@Resource(mappedName = "jms/VisaPagosQueue")  
private static Queue queue;
```

Por otro lado, en el main se ha añadido en el main la inicialización dinámica:

```
InitialContext jndi = new InitialContext();  
  
connectionFactory =  
  
(ConnectionFactory)jndi.lookup("jms/VisaConnectionFactory");  
  
queue = (Queue)jndi.lookup("jms/VisaPagosQueue");
```

La inicialización estática tiene la desventaja de que el nombre se debe conocer la cola en tiempo de compilación.

Ejercicio número 13:

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

Fichero jms.properties se añade el host del servidor que es el de la ip 10.4.6.2, ya que es donde se encuentra el servidor de aplicaciones y los nombres de la factoría y de la cola de mensajes.

```
as.home=${env.J2EE_HOME}  
  
as.lib=${as.home}/lib  
  
as.user=admin  
  
as.host.server=10.4.6.2  
  
as.port=4848  
  
as.passwordfile=${basedir}/passwordfile  
  
as.target=server
```

```
jms.factoryname=jms/VisaConnectionFactory  
jms.name=jms/VisaPagosQueue  
jms.physname=VisaPagosQueue
```

En el fichero build.properties es resaltable la entrada del host.mdb que es el 10.4.6.2, o servidor de aplicaciones:

```
as.host.mdb=10.4.6.2
```

Borre desde la consola de administración de Glassfish la connectionFactory y la cola creada manualmente y ejecute:

```
cd P1-jms
```

```
ant todo
```

```
→ P1-jms git:(master) ✘ ant todo  
Buildfile: /home/eps/Si2/SI2P1B_2311_6/P1-jms-base/P1-jms/build.xml  
  
todo:  
  
setup-jms:  
  
create-jms-connection-factory:  
    [exec] Command create-jms-resource failed.  
    [exec] remote failure: Unable to create connection pool.  
    [exec] A resource named jms/VisaConnectionFactory-Connection-Pool already exists.  
    [exec] Result: 1  
  
create-jms-resource:  
    [exec] Command create-jms-resource failed.  
    [exec] remote failure: Unable to create admin object.  
    [exec] A AdminObjectResource by name jms/VisaPagosQueue already exists with resource-ref in target server.  
    [exec] Result: 1  
  
mdb:  
  
montar-jerarquia:  
  
compilar-mdb:  
  
preparar-meta-inf-mdb:  
  
empaquetar-mdb:  
    [delete] Deleting: /home/eps/Si2/SI2P1B_2311_6/P1-jms-base/P1-jms/dist/mdb/P1-jms-mdb.jar  
    [jar] Building jar: /home/eps/Si2/SI2P1B_2311_6/P1-jms-base/P1-jms/dist/mdb/P1-jms-mdb.jar  
  
desplegar-mdb:  
    [exec] Application deployed with name P1-jms-mdb.  
    [exec] Command deploy executed successfully.  
  
clientjms:  
  
montar-jerarquia:  
  
compilar-clientjms:  
  
empaquetar-clientjms:  
    [delete] Deleting: /home/eps/Si2/SI2P1B_2311_6/P1-jms-base/P1-jms/dist/clientjms/P1-jms-clientjms.jar  
    [jar] Building jar: /home/eps/Si2/SI2P1B_2311_6/P1-jms-base/P1-jms/dist/clientjms/P1-jms-clientjms.jar  
  
BUILD SUCCESSFUL  
Total time: 3 seconds  
→ P1-jms git:(master) ✘ []
```

Compruebe en la consola de administración del Glassfish que, efectivamente, los recursos se han creado automáticamente. Incluye una captura de pantalla, donde se muestre la consola de administración con los recursos creados.

https://10.4.6.2:4848/common/index.jsf

User: admin Domain: domain1 Server: 10.4.6.2

GlassFish™ Server Open Source Edition

Total # of available updates: 1

JMS Connection Factories

Java Message Service (JMS) connection factories are objects that allow an application to create other JMS objects programmatically. Click New to create a new connection factory. Click the name of a connection factory to modify its properties.

Connection Factories (2)					
Select	JNDI Name	Logical JNDI Name	Enabled	Resource Type	Description
<input type="checkbox"/>	jms/_defaultConnectionFactory	java:comp/DefaultJMSConnectionFactory	<input checked="" type="checkbox"/>	javax.jms.ConnectionFactory	
<input type="checkbox"/>	jms/VisaConnectionFactory		<input checked="" type="checkbox"/>	javax.jms.QueueConnectionFactory	

https://10.4.6.2:4848/common/index.jsf

User: admin Domain: domain1 Server: 10.4.6.2

GlassFish™ Server Open Source Edition

Total # of available updates: 1

JMS Destination Resources

JMS destinations serve as the repositories for messages. Click New to create a new destination resource. Click the name of a destination resource to modify its properties.

Destination Resources (1)				
Select	JNDI Name	Enabled	Resource Type	Description
<input type="checkbox"/>	jms/VisaPagosQueue	<input checked="" type="checkbox"/>	javax.jms.Queue	

Revise el fichero jms.xml y anote en la memoria de prácticas cuál es el comando equivalente para crear una cola JMS usando la herramienta asadmin.

```
<target name="create-jms-resource">

    <description>creates jms destination resource</description>

    <exec executable="${asadmin}">

        <arg line="--user ${as.user}" />
        <arg line="--passwordfile ${as.passwordfile}" />
        <arg line="--host ${as.host.server}" />
        <arg line="--port ${as.port}" />
        <arg line="create-jms-resource"/>
        <arg line="--restype ${jms.restype}" />
        <arg line="--enabled=true" />
        <arg line="--property ${jms.resource.property}" />
        <arg line="${jms.resource.name}" />
    </exec>
</target>

<target name="create-jms-connection-factory">

    <description>creates jms connection factory</description>
```

```

<exec executable="${asadmin}">
    <arg line="--user ${as.user}" />
    <arg line="--passwordfile ${as.passwordfile}" />
    <arg line="--host ${as.host.server}" />
    <arg line="--port ${as.port}" />
    <arg line="create-jms-resource"/>
    <arg line="--restype ${jms.restype}" />
    <arg line="--enabled=true" />
    <arg line="${jms.resource.name}" />
</exec>
</target>

```

Ejercicio número 14:

Modifique el cliente, VisaQueueMessageProducer.java, implementando el envío de args[0] como mensaje de texto (consultar los apéndices). Incluye en la memoria el fragmento de código que ha tenido que modificar.

Siguiendo el apartado 8.2.3 y la documentación hemos llegado a las siguientes líneas de código para abrir conexiones, enviar el mensaje y cerrar conexiones:

```

messageProducer = session.createProducer(queue);

message = (TextMessage) session.createTextMessage();

message.setText(args[0]);

messageProducer.send(message);

messageProducer.close();

session.close();

connection.close();

```

Como se puede observar, se crea el productor de mensajes, que es el que realmente enviará los mensajes a través de la cola, se crea un mensaje de texto que será enviado (objeto), y se establece el texto añadido por los argumentos.

A continuación, una vez establecido el texto, el productor envía el mensaje y se cierran todas las conexiones.

Donde 10.X.Y.Z representa la dirección IP de la máquina virtual en cuyo servidor de aplicaciones se encuentra desplegado el MDB. Para garantizar que el comando funcione correctamente es necesario fijar la variable (web console->Configurations->server-config->Java Message Service->JMS Hosts->default_JMS_host) que toma el valor “localhost” por la dirección IP de dicha máquina virtual. El cambio se puede llevar a cabo desde la consola de administración. Será necesario reiniciar el servidor de aplicaciones para que surja efecto.

Será necesario reiniciar el servidor de aplicaciones para que surja efecto.

Se hace un “asadmin stop-domain domain1” y se vuelve a lanzar para aplicar los cambios

Para verificar el contenido de la cola se debe ejecutar: /glassfish/bin/appclient –targetserver 10.X.Y.Z -client dist/clientjms/P1-jms-clientjms.jar -browse

Podemos verificar que la cola de mensajes se ha establecido y se encuentra vacía

```
→ P1-jms git:(master) ✘ /opt/glassfish4/glassfish/bin/appclient -targetserver 10.4.6.2 -client dist/clientjms/P1-jms-clientjms.jar -browse
mar 12, 2022 8:02:54 PM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV000001: Hibernate Validator 5.1.2.Final
Mar 12, 2022 8:02:54 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMACIÓN: MOJMSRA_RA1101: GlassFish MQ JMS Resource Adapter: Version: 5.1.1 (Build 2-c) Compile: March 17 2015 1045
mar 12, 2022 8:02:54 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMACIÓN: MOJMSRA_RA1101: GlassFish MQ JMS Resource Adapter starting: broker is REMOTE, connection mode is TCP
mar 12, 2022 8:02:54 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMACIÓN: MOJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started:REMOTE
Cola de mensajes vacía!
→ P1-jms git:(master) ✘
```

Detenga la ejecución del MDB con la consola de administración para que no consuma los mensajes de la cola (check de ‘Enabled’ en Applications/P1-jms-mdb y guardar los cambios)

Select	Name	Deployment Order	Enabled	Engines	Action
<input type="checkbox"/>	P1-jms-mdb	100	✗	ejb	Redeploy Reload

Envíe un mensaje a la cola, por ejemplo, el texto “idAutorizacion” y compruebe el contenido de la cola.

Incluya en la memoria de prácticas evidencias de todos los pasos, por ejemplo, capturas de pantalla.

Vamos a añadir un 1 de argumentos y además hemos modificado el main para que se imprima en el terminal el mensaje con este código:

```
System.out.println("Mensaje de la cola:" + message.getText());
```

Con lo cual, al ejecutar el programa de nuevo podemos observar el mensaje de la cola:

```
Archivo Editar Ver Buscar Terminal Ayuda
→ P1-jms git:(master) X /opt/glassfish4/glassfish/bin/appclient -targetserver 10.4.6.2 -client dist/clientjms/P1-jms-clientjms.jar 1
mar 12, 2022 8:04:10 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: HV000001: Hibernate Validator 5.1.2.Final
mar 12, 2022 8:04:10 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMACIÓN: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter: Version: 5.1.1 (Build 2-c) Compile: March 17 2015 1045
mar 12, 2022 8:04:10 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMACIÓN: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter starting: broker is REMOTE, connection mode is TCP
mar 12, 2022 8:04:10 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMACIÓN: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started:REMOTE
Mensaje de la cola:1
→ P1-jms git:(master) X
```

A continuación, volver a habilitar la ejecución del MDB y realizar los siguientes pasos:

- Realice un pago con la aplicación web (P1-ejb-transaccional)

En nuestro caso, vamos a realizar el mismo proceso con otro cliente distinto para que se vea que se ha hecho de nuevo:

- Obtenga evidencias de que se ha realizado

Observamos las evidencias en la base de datos, donde el saldo se ha visto decrementado en 25 euros, y se ha añadido un entrada en los pagos:

- Cancélalo con el cliente

Para ello ejecutaremos el comando con el id de Autorización de argumentos = 1 referente al del pago que acabamos de realizar.

```

Archivo Editar Ver Buscar Terminal Ayuda
→ P1-jms git:(master) X /opt/glassfish4/glassfish/bin/appclient -targetserver 10.4.6.2 -client dist/clientjms/P1-jms-clientjms.jar 1
mar 12, 2022 8:30:22 PM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV0000001: Hibernate Validator 5.1.2.Final
mar 12, 2022 8:30:22 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMACIÓN: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter: Version: 5.1.1 (Build 2-c) Compile: March 17 2015 1045
mar 12, 2022 8:30:22 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMACIÓN: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter starting: broker is REMOTE, connection mode is TCP
mar 12, 2022 8:30:22 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMACIÓN: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started:REMOTE
Mensaje de la cola:1

```

- Obtenga evidencias de que se ha cancelado y de que el saldo se ha rectificado

Ahora vamos a observar las evidencias en la base de datos, para ello, recordamos que primero debemos observar que el código de verificación se ha puesto en 999 referente a que ha sido cancelado por el cliente (luego no podrá volver a cancelarlo). Y posteriormente, el saldo se ha restablecido a 1000 del cliente, y efectivamente observamos que ha sido exitosa la cancelación:

	idautorizacion	accidtransaccion	acccodresposta	accidcomercio	accnumerotarjeta	fecha
1	1	999	25	1	1111 2222 3333 4444	2022-03-12 11:22:45.775

	numerotarjeta	titular	validadesde	validahasta	codigoverificacion	saldo
1	1111 2222 3333 4444	Jose Garcia	11/09	11/22	123	1.000

● CUESTIONES:

- Cuestión 1: Abrir el archivo VisaDAOLocal.java y comprobar la definición de dicha interfaz. Anote en la memoria comentarios sobre las librerías Java EE importadas y las anotaciones utilizadas. ¿Para qué se utilizan? Comparar esta interfaz con el fichero de configuración del web service implementado en la práctica P1A.**

Se hace uso de la librería javax.ejb.local, para hacer uso de la anotación @local, que se aplica en la interfaz de "visaDaoLocal" con todos los métodos que probablemente tenga que implementar "visaDAOBean".

Esta anotación @local, lo que indica, es que efectivamente, el cliente y el servidor se van a ejecutar en una misma máquina, con lo cual solo habrá un servidor Glassfish para esta primera iteración.

Las etiquetas implementadas en la anterior práctica corresponden a implementaciones de "VisaDAO" que se iba a usar como servicio WEB accesible por el cliente a través del navegador.

- **Cuestión 2: Abrir el archivo application.xml y explicar su contenido. Verifique el contenido de todos los archivos .jar / .war / .ear que se han construido hasta el momento (empleando el comando jar –tvf). Explique brevemente el contenido y ponga evidencias en la memoria.**

En el fichero se hace una representación en el estándar XML de los distintos módulos que serán desplegados de la aplicación, con las etiquetas <module>, uno para el fichero .jar de la aplicación servidora generada, y otro para el acceso web al recurso <web-uri> .war del cliente, y la ruta necesaria para acceder a él <context-root> /P1-ejb-cliente.

Si consultamos los archivos del .war del cliente se obtiene que está compuesto por todos los elementos relacionados con controladores y vistas de la aplicación que serán las importantes para el cliente .

```

Archivo Editar Ver Buscar Terminal Ayuda
→ ~ jar -tvf SIST2/P1B/P1-ejb/dist/client/P1-ejb-cliente.war
  0 Sat Mar 05 21:02:50 CET 2022 META-INF/
125 Sat Mar 05 21:02:48 CET 2022 META-INF/MANIFEST.MF
  0 Sat Mar 05 21:02:50 CET 2022 WEB-INF/
  0 Sat Mar 05 20:55:50 CET 2022 WEB-INF/classes/
  0 Sat Mar 05 21:01:48 CET 2022 WEB-INF/classes/ssil2/
  0 Sat Mar 05 21:01:48 CET 2022 WEB-INF/classes/ssil2/controlador/
  0 Sat Mar 05 21:01:48 CET 2022 WEB-INF/classes/ssil2/filtros/
  0 Sat Mar 05 21:01:48 CET 2022 WEB-INF/classes/ssil2/visa/
  0 Sat Mar 05 21:01:48 CET 2022 WEB-INF/classes/ssil2/visa/error/
  0 Sat Mar 05 20:55:38 CET 2022 WEB-INF/lib/
  0 Sat Mar 05 21:02:50 CET 2022 error/
2844 Sat Mar 05 20:55:50 CET 2022 WEB-INF/classes/ssil2/controlador/ComienzaPago.class
1518 Sat Mar 05 20:58:18 CET 2022 WEB-INF/classes/ssil2/controlador/DelPagos.class
1370 Sat Mar 05 21:01:48 CET 2022 WEB-INF/classes/ssil2/controlador/GetPagos.class
4929 Sat Mar 05 21:01:48 CET 2022 WEB-INF/classes/ssil2/controlador/ProcesaPago.class
1894 Sat Mar 05 20:55:50 CET 2022 WEB-INF/classes/ssil2/controlador/ServletRatz.class
2608 Sat Mar 05 21:01:48 CET 2022 WEB-INF/classes/ssil2/filtros/CompruebaSession.class
3170 Sat Mar 05 21:01:48 CET 2022 WEB-INF/classes/ssil2/visa/ValidadorTarjeta.class
616 Sat Mar 05 21:01:48 CET 2022 WEB-INF/classes/ssil2/visa/error/ErrorVisa.class
198 Sat Mar 05 21:01:48 CET 2022 WEB-INF/classes/ssil2/visa/error/ErrorVisaCV.class
209 Sat Mar 05 21:01:48 CET 2022 WEB-INF/classes/ssil2/visa/error/ErrorVisaFechaCaducidad.class
207 Sat Mar 05 21:01:48 CET 2022 WEB-INF/classes/ssil2/visa/error/ErrorVisaFechaExpiracion.class
201 Sat Mar 05 21:01:48 CET 2022 WEB-INF/classes/ssil2/visa/error/ErrorVisaNumero.class
202 Sat Mar 05 21:01:48 CET 2022 WEB-INF/classes/ssil2/visa/error/ErrorVisaTitular.class
6024 Sat Mar 05 21:02:50 CET 2022 WEB-INF/web.xml
455 Sat Mar 05 21:02:50 CET 2022 borradoerror.jsp
501 Sat Mar 05 21:02:50 CET 2022 borradook.jsp
509 Sat Mar 05 21:02:50 CET 2022 cabecera.jsp
283 Sat Mar 05 21:02:50 CET 2022 error/nuestraerror.jsp
2729 Sat Mar 05 21:02:50 CET 2022 formdatosvisa.jsp
1257 Sat Mar 05 21:02:50 CET 2022 listapagos.jsp
1178 Sat Mar 05 21:02:50 CET 2022 pago.html
1142 Sat Mar 05 21:02:50 CET 2022 pagoexito.jsp
104 Sat Mar 05 21:02:50 CET 2022 ple.html
5011 Sat Mar 05 21:02:50 CET 2022 testbd.jsp

```

En cuanto al .jar de la aplicación servidora obtenemos que está formada por las clases que se encargan del backend de la aplicación como acceso a base de datos siguiendo el modelo VISADAO.

```

Archivo Editar Ver Buscar Terminal Ayuda
→ ~ jar -tvf SIST2/P1B/P1-ejb/dist/server/P1-ejb.jar
  0 Sat Mar 05 20:55:42 CET 2022 META-INF/
125 Sat Mar 05 20:55:40 CET 2022 META-INF/MANIFEST.MF
  0 Sat Mar 05 20:55:40 CET 2022 ssil2/
  0 Sat Mar 05 20:55:40 CET 2022 ssil2/visa/
235 Sat Mar 05 20:55:42 CET 2022 META-INF/sun-ejb-jar.xml
1719 Sat Mar 05 20:55:40 CET 2022 ssil2/visa/DBTester.class
1464 Sat Mar 05 20:55:40 CET 2022 ssil2/visa/PagoBean.class
856 Sat Mar 05 20:55:40 CET 2022 ssil2/visa/TarjetaBean.class
7031 Sat Mar 05 20:55:40 CET 2022 ssil2/visa/VisaDAOBean.class
593 Sat Mar 05 20:55:40 CET 2022 ssil2/visa/VisaDAOLocal.class

```

Finalmente, el fichero P1-ejb.ear obtenido de empaquetar la aplicación, contiene las especificaciones comentadas anteriormente en el fichero application.xml, y el .jar y .war a los que hace referencia la misma.

```

Archivo Editar Ver Buscar Terminal Ayuda
→ ~ jar -tvf SIST2/P1B/P1-ejb/dist/P1-ejb.ear
  0 Sun Mar 06 13:32:18 CET 2022 META-INF/
125 Sun Mar 06 13:32:16 CET 2022 META-INF/MANIFEST.MF
508 Sun Mar 06 13:09:56 CET 2022 META-INF/application.xml
29977 Sat Mar 05 21:02:50 CET 2022 P1-ejb-cliente.war
6889 Sat Mar 05 20:55:42 CET 2022 P1-ejb.jar

```