



PRÁCTICA 1A

Sistemas Informáticos 2

Adrián San Felipe y Luis Miguel Durán

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.

Incluya en la memoria de prácticas todos los pasos necesarios para resolver este ejercicio así como las evidencias obtenidas. Se pueden incluir por ejemplo capturas de pantalla.

Para desplegar la aplicación en Glassfish, primero tenemos que modificar ciertos parámetros de los archivos **build.properties** y **postgresql.properties** que se encuentran en la carpeta P1-base. En concreto, debemos añadir la dirección IP asignada en nuestras prácticas con nuestro grupo de usuario en las siguientes líneas de los archivos:

```
build.properties
~/Escritorio/P1-base
Guardar

# Propiedades de despliegue de aplicacion de Visa
nombre=P1
build=${basedir}/build

dist=${basedir}/dist

src=${basedir}/src

web=${basedir}/web

paquete=ssli2
war=${nombre}.war

asadmin=${as.home}/bin/asadmin
as.home=${env.J2EE_HOME}
as.lib=${as.home}/lib
as.user=admin
as.host=10.5.9.1
as.port=4848
as.passwordfile=${basedir}/passwordfile
as.target=server
```

Texto plano Anchura del tabulador: 8 Ln 29, Col 1 INS

```
postgresql.properties
~/Escritorio/P1-base
Guardar

# Propiedades de la BD postgresql

# Parametros propios de postgresql
db.name=visa
db.user=alumnodb
db.password=****
db.port=5432
db.host=10.5.9.1
# Recursos y pools asociados
db.pool.name=VisaPool
db.jdbc.resource.name=jdbc/VisaDB
db.url=jdbc:postgresql://${db.host}:${db.port}/${db.name}
db.client.host=10.5.9.1
db.client.port=4848

db.delimiter=;
db.driver=org.postgresql.Driver
db.datasource=org.postgresql.ds.PGConnectionPoolDataSource
db.vendorname=SQL92

# Herramientas
db.createdb=/usr/bin/createdb
db.dropdb=/usr/bin/dropdb

# Scripts de creacion / borrado
db.create.src=./sql/create.sql
db.insert.src=./sql/insert.sql
db.delete.src=./sql/drop.sql
```

Texto plano Anchura del tabulador: 8 Ln 28, Col 29 INS

Después de cambiar dichos archivos, tenemos que desplegar la aplicación con el comando “ant todo”, si la compilación y el despliegue se ha realizado correctamente, nos deberá aparecer un mensaje así:

```
BUILD SUCCESSFUL
Total time: 1 minute 34 seconds
eps@labvirtips:~/Escritorio/P1-base$ ant todo
```

Ahora, podemos ver desde Glassfish como la aplicación está desplegada:

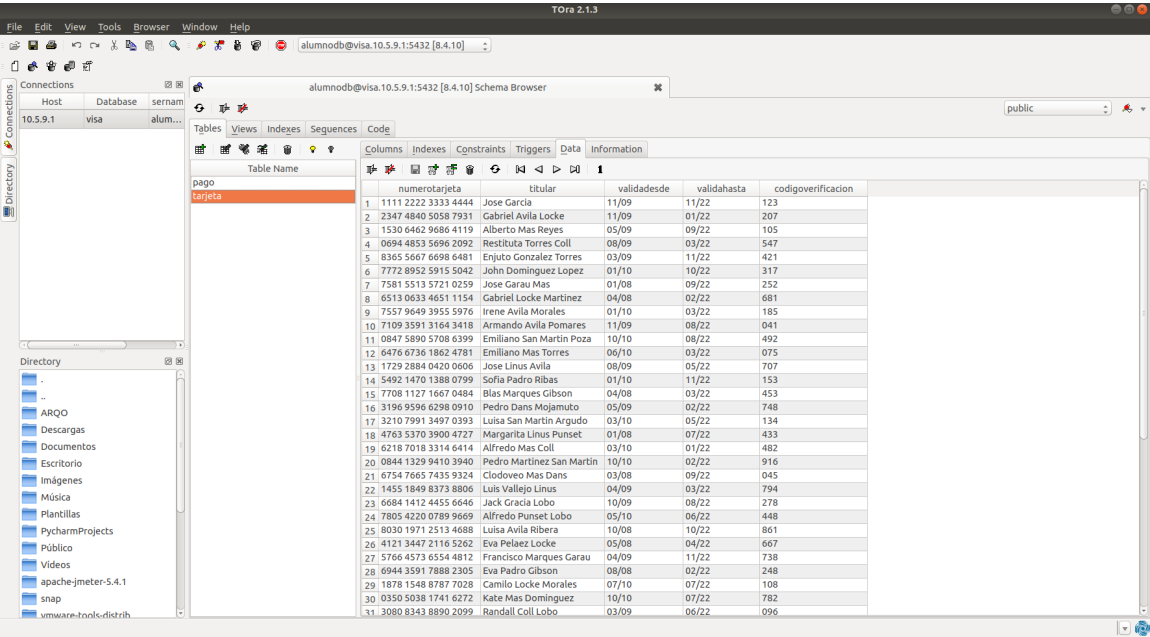
Select	Name	Deployment Order	Enabled	Engines	Action
<input type="checkbox"/>	P1	100	✓	web	Launch Redeploy Reload

A continuación, desde Tora podemos ver la base de datos desplegada anteriormente, y comprobar como efectivamente no existe ningún pago.

Table Name
pago
tarjeta

Columns
idautorizacion
idtransaccion
codrespuesta
importe
idcomercio
numerotarjeta
fecha

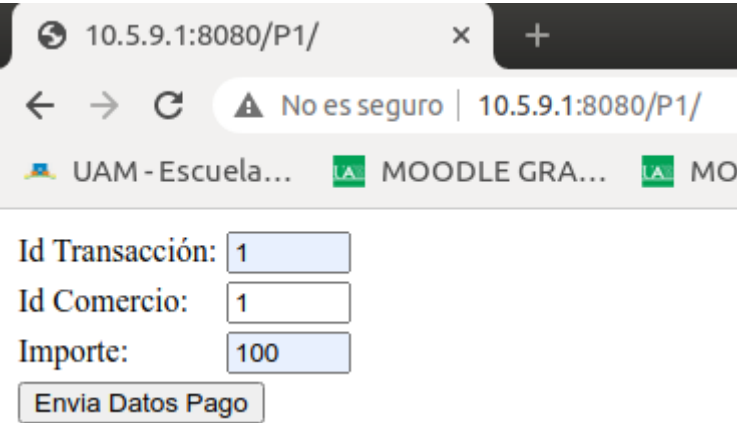
Antes de nada, necesitaremos ver desde Tora los datos de las tarjetas para realizar el pago en la siguiente página que se nos mostrará, e introducimos dichos datos en la página:



The screenshot shows the Tora 2.1.3 interface with the 'alumnodb@visa.10.5.9.1:5432 [8.4.10]' database selected. The 'tarjeta' table is highlighted in the left pane. The main pane displays the table's data with columns: numerotarjeta, titular, validadesde, validahasta, and codigoverificacion. The data is as follows:

	numerotarjeta	titular	validadesde	validahasta	codigoverificacion
1	1111 2222 3333 4444	Jose Garcia	11/09	11/22	123
2	2347 4840 5058 7931	Gabriel Avila Locke	11/09	01/22	207
3	1530 6462 9686 4119	Alberto Mas Reyes	05/09	09/22	105
4	0694 4853 5696 2092	Restituta Torres Coll	08/09	03/22	547
5	8365 5667 6698 6481	Enjuto Gonzalez Torres	03/09	11/22	421
6	7772 8952 5915 5042	John Dominguez Lopez	01/10	10/22	317
7	7581 5513 5721 0259	Jose Garau Mas	01/08	09/22	252
8	6513 0633 4651 1154	Gabriel Locke Martinez	04/08	02/22	681
9	7537 9649 3955 5976	Irene Avila Morales	01/10	03/22	185
10	7199 3591 3164 3418	Armando Avila Pomares	11/09	08/22	041
11	0847 5890 5708 4399	Emiliano San Martin Poza	10/10	08/22	492
12	6476 6736 1862 4781	Emiliano Mas Torres	06/10	03/22	075
13	1729 2884 0420 0606	Jose Linus Avila	08/09	05/22	707
14	5492 1470 1388 0799	Sofia Padro Ribas	01/10	11/22	153
15	7708 1127 1667 0484	Blas Marques Gibson	04/08	03/22	453
16	3196 9596 6298 0910	Pedro Dans Mojamuto	05/09	02/22	748
17	3210 7991 3497 0393	Luisa San Martin Argudo	03/10	05/22	134
18	4763 5370 3900 4727	Margarita Linus Punset	01/08	07/22	433
19	6218 7018 3314 6414	Alfredo Mas Coll	03/10	01/22	482
20	0844 1329 9410 3940	Pedro Martinez San Martin	10/10	02/22	916
21	6754 7665 7435 9324	Clodoveo Mas Dans	03/08	09/22	045
22	1455 1849 8373 8806	Luis Vallejo Linus	04/09	03/22	794
23	6684 1412 4455 6646	Jack Gracia Lobo	10/09	08/22	278
24	7805 4220 0789 9669	Alfredo Punset Lobo	05/10	06/22	448
25	8030 1971 2513 4688	Luisa Avila Ribera	10/08	10/22	861
26	4121 3447 2116 5262	Eva Pelaez Locke	05/08	04/22	667
27	5766 4573 6554 4812	Francisco Marques Garau	04/09	11/22	738
28	6944 3591 7888 2305	Eva Padro Gibson	08/08	02/22	248
29	1878 1548 8787 7028	Camilo Locke Morales	07/10	07/22	108
30	0369 5038 1741 6272	Kate Mas Dominguez	10/10	07/22	782
31	3080 8343 8890 2099	Randall Coll Lobo	03/09	06/22	096

Accedemos a la página de pago e introducimos los datos que queramos:



The screenshot shows a web browser window with the address bar displaying '10.5.9.1:8080/P1/'. The page title is 'UAM - Escuela...'. The main content area contains a payment form with the following fields:

Id Transacción:

Id Comercio:

Importe:

Envia Datos Pago (button)

Sistema de Pago con tarjeta x +

← → ↺ No es seguro | 10.5.9.1:8080/P1/comienzapago

UAM - Escuela... MOODLE GRA... MOODLE POS...

Pago con tarjeta

Numero de visa:

Titular:

Fecha Emisión:

Fecha Caducidad:

CVV2:

Id Transacción: 1
Id Comercion: 1
Importe: 100.0

Prácticas de Sistemas Informáticos II

Podemos observar que el pago se ha realizado correctamente:

Pago con tarjeta

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

idTransaccion: 1
idComercio: 1
importe: 100.0
codRespuesta: 000
idAutorizacion: 1

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

Accedemos a la base de datos desde Tora y podemos observar como el pago ha quedado reflejado en la base de datos:

Tora 2.1.3

File Edit View Tools Browser Window Help

alumnodb@visa.10.5.9.1:5432 [8.4.10]

alumnodb@visa.10.5.9.1:5432 [8.4.10] SQL Editor - Untitled

alumnodb@visa.10.5.9.1:5432 [8.4.10] Schema Browser

public

Table Name	Columns	Indexes	Constraints	Triggers	Data	Information
pago	idautorizacion	idtransaccion	codrespuesta	importe	idcomercio	numerotarjeta
tarjeta	1	1	000	100	1	1111 2222 3333 4444
						28/02/21 07:27

Directory

-
- ARQO
- Descargas
- Documentos
- Escritorio
- Imágenes
- Música
- Plantillas

Volvemos a la página anterior para consultar y borrar el pago que hemos realizado:

Pago con tarjeta

Consulta de pagos

Lista de pagos del comercio 1

Id Comercio:

idTransaccion	Importe	codRespuesta	idAutorizacion
1	100.0	000	1

[Volver al comercio](#)

Borrado de pagos

Id Comercio:

Pago con tarjeta

Se han borrado 1 pagos correctamente para el comercio 1

[Volver al comercio](#)

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 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 que funcione la conexión directa deberemos cambiar el valor de ciertas variables que se encuentran en el archivo `DBTester.java`, dicho archivo se encuentra en la ruta `src/ssii2/visa/dao`.

Debemos dejar el archivo tal y como se encuentra en la siguiente imagen:

```
public class DBTester {

    // Información de conexión
    // Para conexiones directas, requerimos: driver, cadena de conexión,
    // usuario y clave
    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://10.5.9.1:5432/visa";
    *****/
    private static final String JDBC_USER = "alumnodb";
    private static final String JDBC_PASSWORD = "alumnodb";

    // Para conexión por datasource, sólo necesitamos su nombre
    // TODO: Definir el nombre del datasource
    /*****
    private static final String JDBC_DSN =
        "jdbc/VisaDB";
    *****/

    // Modo inicial de conexión (directo|dsn)
    private boolean directConnection = false;

    // Datasource para conexiones por pool
    private DataSource ds = null;

    // Información de debug
    private int dsConnectionCount = 0;
    private int directConnectionCount = 0;
```

Hemos modificado las variables JDBC_DRIVER, JDBC_CONNSTRING, JDBC_USER y JDBC_PASSWORD.

Ahora, volvemos a la página anterior y marcamos la opción de Direct Connection.

Pago con tarjeta

Proceso de un pago

Id Transacción:

Id Comercio:

Importe:

Numero de visa:

Titular:

Fecha Emisión:

Fecha Caducidad:

CVV2:

Modo debug: ☐ True ☐ False

Direct Connection: ☒ True ☐ False

Use Prepared: ☐ True ☐ False

Como podemos observar, el pago se ha realizado con éxito con conexión directa.

Pago con tarjeta

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

idTransaccion: 2
idComercio: 2
importe: 200.0
codRespuesta: 000
idAutorizacion: 2

[Volver al comercio](#)

Pago con tarjeta

Consulta de pagos

Lista de pagos del comercio 2

Id Comercio:

idTransaccion	Importe	codRespuesta	idAutorizacion
2	200.0	000	2

[Volver al comercio](#)

Borrado de pagos

Id Comercio:

Pago con tarjeta

Se han borrado 1 pagos correctamente para el comercio 2

[Volver al comercio](#)

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.

```
# Propiedades de la BD postgresql

# Parametros propios de postgresql
db.name=visa
db.user=alumnodb
db.password=****
db.port=5432
db.host=10.5.9.1
# Recursos y pools asociados
db.pool.name=VisaPool
db.jdbc.resource.name=jdbc/VisaDB
db.url=jdbc:postgresql://${db.host}:${db.port}/${db.name}
db.client.host=10.5.9.1
db.client.port=4848

db.delimiter=;
db.driver=org.postgresql.Driver
db.datasource=org.postgresql.ds.PGConnectionPoolDataSource
db.vendorname=SQL92

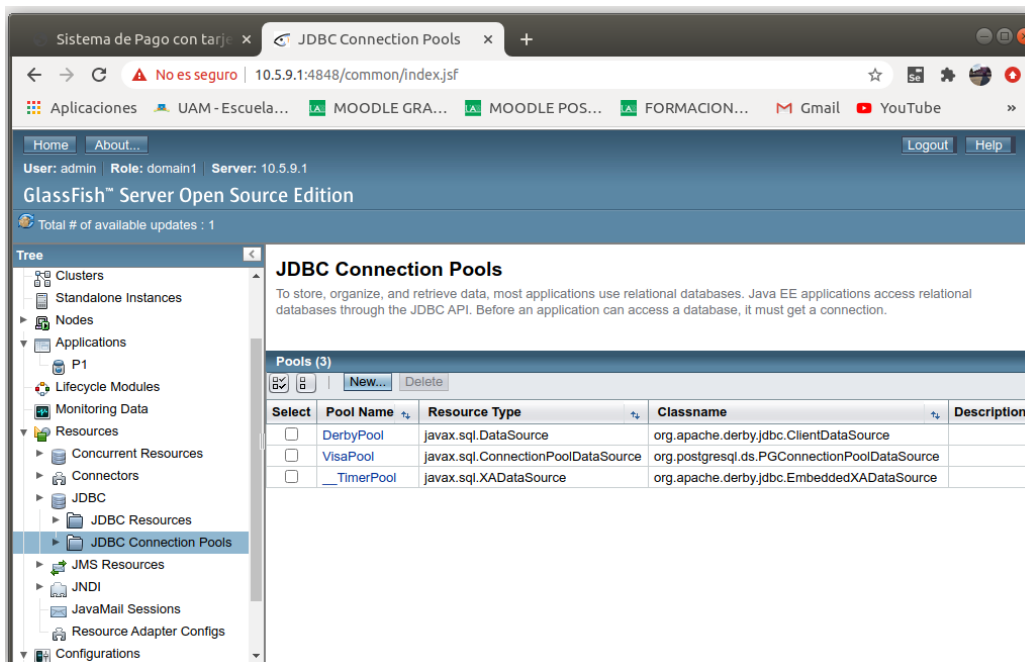
# Herramientas
db.createdb=/usr/bin/createdb
db.dropdb=/usr/bin/dropdb

# Scripts de creacion / borrado
db.create.src=./sql/create.sql
db.insert.src=./sql/insert.sql
db.delete.src=./sql/drop.sql
```

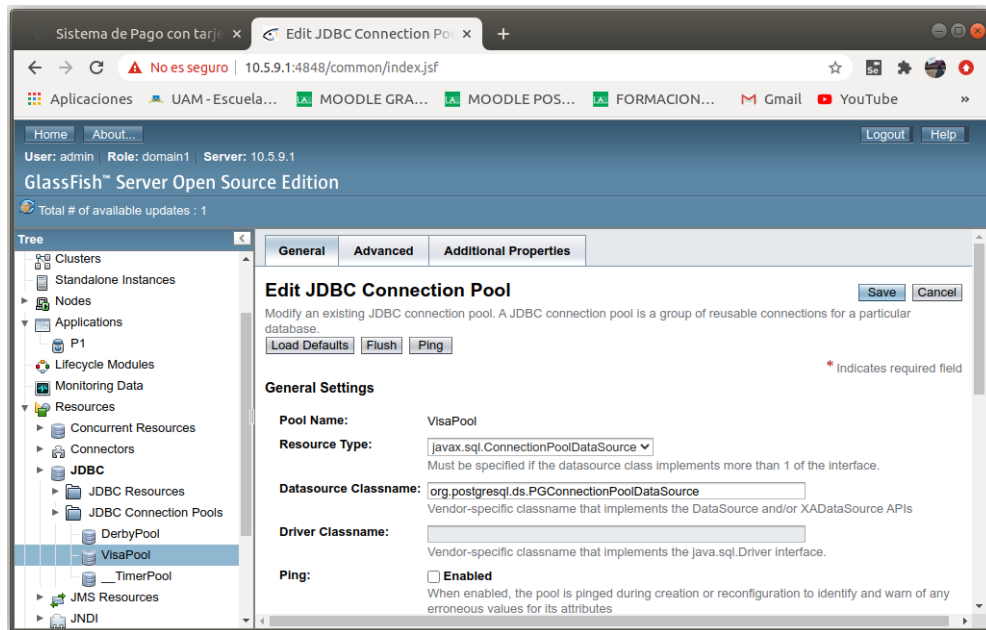
La anterior imagen es el contenido en su totalidad del archivo *postgresql.properties*

De los datos podemos observar que el nombre del recurso JDBC y el nombre del pool aparecen en las variables *db.pool.name* y *db.jdbc.resource.name*

Ahora, accediendo a la consola de administración de Glassfish, podemos ver que los recursos necesarios han sido correctamente creados:



Para hacer el ping, deberemos meternos dentro de VisaPool y pulsar en el botón indicado como “Ping”:



General Advanced Additional Properties

✓ Ping Succeeded

Edit JDBC Connection Pool [Save] [Cancel]

Modify an existing JDBC connection pool. A JDBC connection pool is a group of reusable connections for a particular database.

[Load Defaults] [Flush] [Ping]

* Indicates required field

General Settings

Pool Name: VisaPool

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

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

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

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

Deployment Order: 100
Specifies the loading order of the resource at server startup. Lower numbers are loaded first.

Description:

Pool Settings

Initial and Minimum Pool Size: 8 Connections
Minimum and initial number of connections maintained in the pool

Maximum Pool Size: 32 Connections
Maximum number of connections that can be created to satisfy client requests

Pool Resize Quantity: 2 Connections
Number of connections to be removed when pool idle timeout expires

Idle Timeout: 300 Seconds
Maximum time that connection can remain idle in the pool

Max Wait Time: 60000 Milliseconds
Amount of time caller waits before connection timeout is sent

Si observamos la imagen anterior, podemos averiguar una serie de datos:

- Tamaño inicial y mínimo del Pool: 8 conexiones
- Tamaño máximo del Pool: 32 conexiones
- Cantidad de reajustes de tamaño del Pool: 2 conexiones
- Tiempo de inactividad: 300 segundos
- Tiempo máximo de espera: 60000 milisegundos

Cuanto mayor fuese el tamaño inicial del pool, más lento sería ya que habría más conexiones, de la misma forma que afectaría al tamaño máximo del pool. Si el tiempo de inactividad es muy bajo, podría no dejar tiempo a las respuestas del servidor y provocar saturación. Lo mismo pasa con el tiempo máximo de espera, cuanto más se aumentase, más saturación podría provocar.

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.

Los fragmentos de código se encuentran en el archivo VisaDAO.java en la ruta src/ssii2/visa/dao

El código es el siguiente:

```
/**
 * getQryCompruebaTarjeta
 */
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;
}

/**
 * getQryInsertPago
 */
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 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 una captura de pantalla del log del servidor.

```
// TODO: Utilizar en funcion de isPrepared()
PreparedStatement pstmt = null;

try {

    // Crear una conexion u obtenerla del pool
    con = getConnection();

    // Se busca la ocurrencia de la tarjeta en la tabla

    /* TODO Usar prepared statement si
    isPrepared() == true */
    /*****
    if (isPrepared() == true) {
        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();

    } else {
        /*****
        stmt = con.createStatement();
        qry = getQryCompruebaTarjeta(tarjeta);
        errorLog(qry);
        rs = stmt.executeQuery(qry);

    } /*****/

    /* Si hay siguiente registro, la tarjeta valido OK */
    ret = rs.next();

} catch (Exception ee) {
    errorLog(ee.toString());
    ret = false;
} finally {
    try {
        if (rs != null) {
            rs.close(); rs = null;
        }
    }
}
```

Podemos observar que errorLog() se encuentra en varias partes del código de VisaDAO.java (se encuentra en más partes de las señaladas)

Ahora, realizamos de nuevo un pago con la opción debug activada:

Pago con tarjeta

Proceso de un pago

Id Transacción:	<input type="text" value="1"/>
Id Comercio:	<input type="text" value="1"/>
Importe:	<input type="text" value="100"/>
Numero de visa:	<input type="text" value="1111 2222 3333 4444"/>
Titular:	<input type="text" value="Jose Garcia"/>
Fecha Emisión:	<input type="text" value="11/09"/>
Fecha Caducidad:	<input type="text" value="11/22"/>
CVV2:	<input type="text" value="123"/>
Modo debug:	<input checked="" type="radio"/> True <input type="radio"/> False
Direct Connection:	<input type="radio"/> True <input type="radio"/> False
Use Prepared:	<input type="radio"/> True <input type="radio"/> False
<input type="button" value="Pagar"/>	

Y en los logs, podemos ver como el proceso del pago aparece reflejado:

Log Viewer Results (40)		
Records before 416	Log File Record Numbers 416 through 455	Records after 455
Record Number	Log Level	Message
455	SEVERE	[directConnection=false] select idAutorizacion, codRespuesta from pago where idTransaccion = '1' ... (details)
454	SEVERE	[directConnection=false] insert into pago(idTransaccion,importe,idComercio,numeroTarjeta) values ('1... (details)
453	SEVERE	[directConnection=false] select * from tarjeta where numeroTarjeta='1111 2222 3333 4444' and titular... (details)

Log Entry Detail

Timestamp	28-feb-2021 09:21:56.292
Log Level	SEVERE
Logger	
Name-Value Pairs	{levelValue=1000, timeMillis=1614532916292}
Record Number	454
Message ID	
Complete Message	[directConnection=false] insert into pago(idTransaccion,importe,idComercio,numeroTarjeta) values ('1',100.0,'1','1111 2222 3333 4444')

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()`

De la clase `DBTester`, de la que hereda `VisaDAOWS.java`, deberemos publicar así mismo:

- `isDirectConnection()` / `setDirectConnection()`

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:

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

Para implementar de manera correcta el servicio Web debemos introducir los siguientes cambios en el archivo `VisaDAOWS.java`:

- Añadir `@WebService()` antes de la clase

```
/**
 * @author jaime
 */
@WebService()
public class VisaDAOWS extends DBTester {

    • compruebaTarjeta()

    /**
     * Comprobacion de la tarjeta
     * @param tarjeta Objeto con toda la informacion de la tarjeta
     * @return true si la comprobacion contra las tarjetas contenidas en
     *         en la tabla TARJETA fue satisfactoria, false en caso contrario */
     @WebMethod(operationName = "compruebaTarjeta")
    public boolean compruebaTarjeta(@WebParam(name = "tarjeta")TarjetaBean tarjeta) {
```

- `realizaPago()`

```
/**
 * Realiza el pago
 * @param pago
 * @return
 */
 @WebMethod(operationName = "realizaPago")
 public synchronized PagoBean realizaPago(@WebParam(name = "pago")PagoBean pago) {
```

- isDebug()

```
/**
 * @return the debug
 */
@WebMethod(operationName = "isDebug")
public boolean isDebug() {
    return debug;
}
```

- setDebug()

Como se nos indica en la práctica, existen dos métodos setDebug, pero con exclude=true evitamos la publicación de uno de ellos.

```
/**
 * @param debug the debug to set
 */
@WebMethod(operationName = "setDebug")
public void setDebug(@WebParam(name = "debug") boolean debug) {
    this.debug = debug;
}

/**
 * @param debug the debug to set
 */
@WebMethod(exclude=true)
public void setDebug(String debug) {
    this.debug = (debug.equals("true"));
}
```

- isPrepared()

```
@WebMethod(operationName = "isPrepared")
public boolean isPrepared() {
    return prepared;
}
```

- setPrepared()

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

- isDirectConnection()

```
/**
 * @return the pooled
 */
@WebMethod(operationName = "isDirectConnection")
@Override
public boolean isDirectConnection() {
    return super.isDirectConnection();
}
```

- setDirectConnection()

```
/**
 * @param directConnection valor de conexión directa o indirecta
 */
@WebMethod(operationName = "setDirectConnection")
@Override
public void setDirectConnection(@WebParam(name = "directConnection") boolean directConnection) {
    super.setDirectConnection(directConnection);
}
```

En realizaPago() hemos añadido inicializar la variable pago a null cuando no se puede realizar el pago y que devuelva la misma en vez de ret:

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

} catch (Exception e) {
    errorLog(e.toString());
    ret = false;
    pago = null;
} finally {
    try {
        if (rs != null) {
            rs.close(); rs = null;
        }
        if (stmt != null) {
            stmt.close(); stmt = null;
        }
        if (pstmt != null) {
            pstmt.close(); pstmt = null;
        }
        if (con != null) {
            closeConnection(con); con = null;
        }
    } catch (SQLException e) {
    }
}

return pago;
}
```

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

Porque si no, no podríamos obtener acceso a los datos del pago.

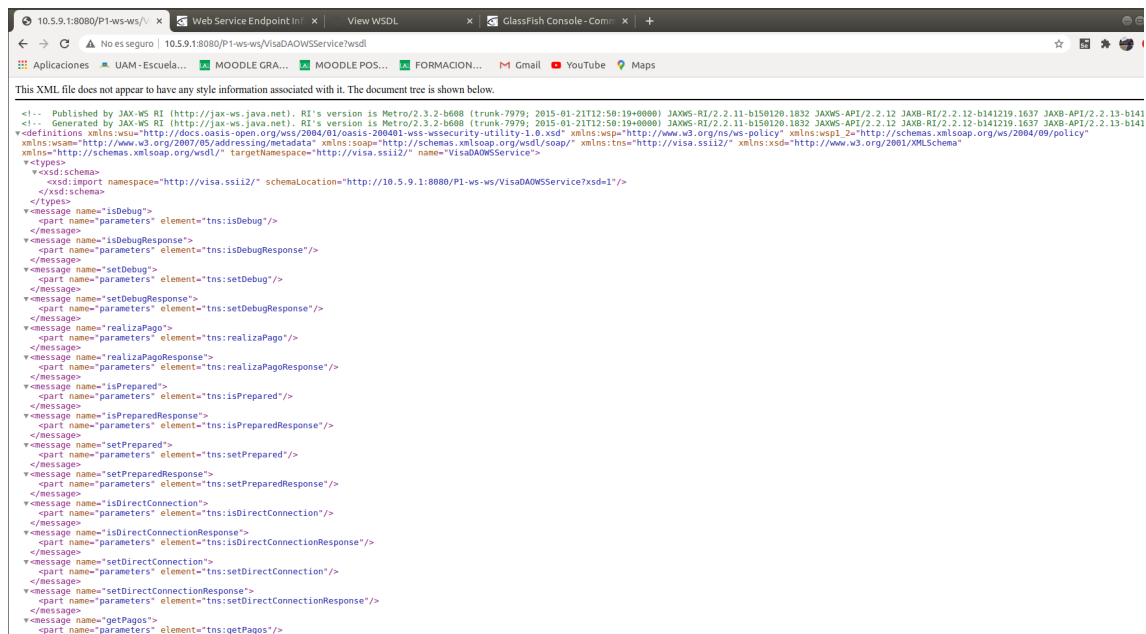
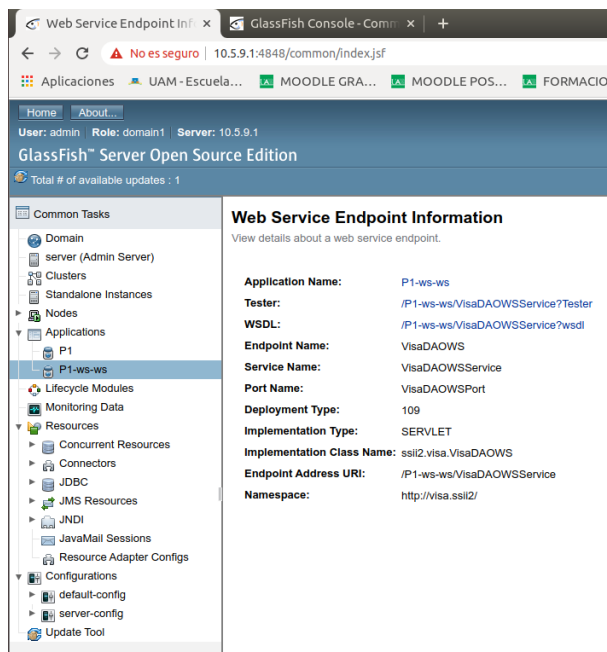
Ejercicio 7. Despliegue el servicio con la regla correspondiente en el *build.xml*. Acceda al WSDL remotamente con el navegador e inclúyalo en la memoria de la práctica (habrá que asegurarse que la URL contiene la dirección IP de la máquina virtual donde se encuentra el servidor de aplicaciones).

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?
- ¿Qué tipos de datos predefinidos se usan?
- ¿Cuáles son los tipos de datos que se definen?
- ¿Qué etiqueta está asociada a los métodos invocados en el webservice?
- ¿Qué etiqueta describe los mensajes intercambiados en la invocación de los métodos del webservice?
- ¿En qué etiqueta se especifica el protocolo de comunicación con el webservice?
- ¿En qué etiqueta se especifica la URL a la que se deberá conectar un cliente para acceder al webservice?

Una vez desplegado podemos observar cómo aparece la aplicación P1-ws-ws y su información, como indica en el enunciado/guion.



Preguntas:

- Los datos se encuentran en el fichero wsdl, como podemos ver en la imagen anterior.

```
<definitions xmlns:wsu="http://docs.oasis-open.org/ws/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns:wsp="http://www.w3.org/ns/ws-policy" xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://visa.ssi12/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://visa.ssi12/" name="VisaDAOWSService">
```

- Los datos xsd:schema
- Todas las etiquetas <message>
- Las etiquetas <portType>
- Las etiquetas <binding>
- La etiqueta <service>

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.

Añadimos las siguientes líneas de código donde se encuentran los imports en `ProcesaPago.java`:

```
import ssi12.visa.VisaDAOWSService; // Stub generado automáticamente
import ssi12.visa.VisaDAOWS; // Stub generado automáticamente
import javax.xml.ws.WebServiceRef;
```

Nuevo método de realiza pago (ya que ahora no devuelve bool, sino pago):

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

Llamadas remotas pueden generar excepciones:

```
/**
 * Procesa una petici&oacute;n HTTP tanto <code>GET</code> como <code>POST</code>.
 * @param request objeto de petici&oacute;n
 * @param response objeto de respuesta
 */
@Override
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
```

Ejercicio 9. Modifique la llamada al servicio para que la ruta 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.

El fichero `web.xml` se encuentra en la ruta `web/WEB-INF`.

En este, editamos y descomentamos la etiqueta <context-param> para que nos quede así:

```
<context-param>
  <param-name>webmaster</param-name>
  <param-value>http://10.5.9.1:8080/P1-ws-ws/VisaDAOWSService</param-value>
</context-param>
```

En la función `processRequest` modificamos la línea “`VisaDAO dao = new VisaDAO();`” por los siguiente:

```
VisaDAOWSService service = new VisaDAOWSService();
VisaDAOWS dao = service.getVisaDAOWSPort ();
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<PagoBean> 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.

En GetPagos.java:

```
/* Petición de los pagos para el comercio */
PagoBean[] pagos = (PagoBean[] ) dao.getPagos(idComercio).toArray(new PagoBean[dao.getPagos(idComercio).size()]);
```

En VisaDAOWS.java, cambiamos esto:

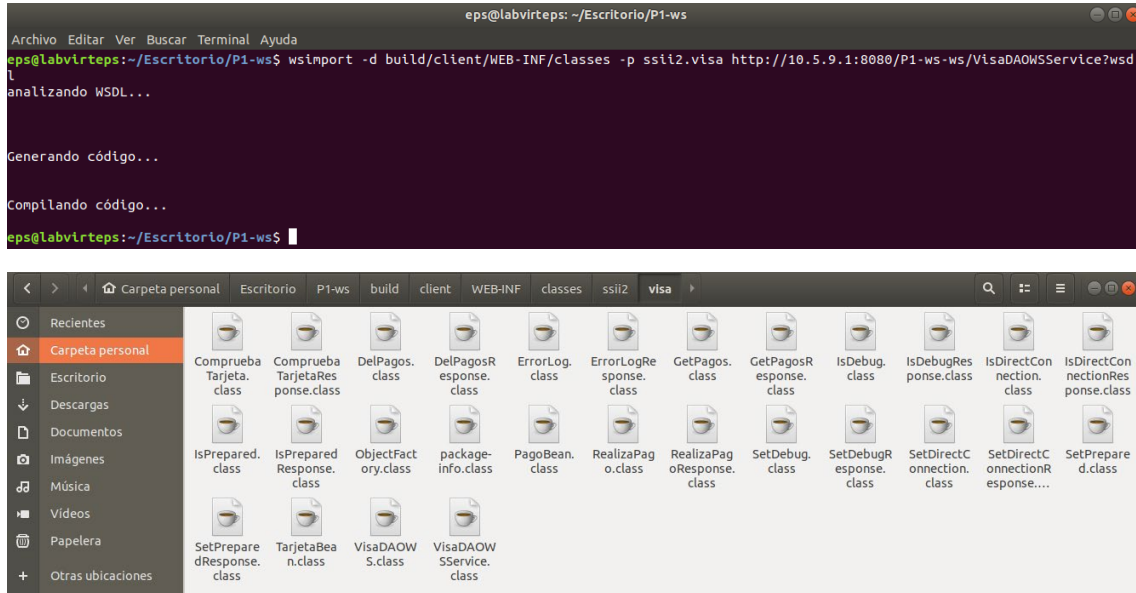
```
/**
 * Buscar los pagos asociados a un comercio
 * @param idComercio
 * @return
 */
public PagoBean[] getPagos(String idComercio) {
```

A esto:

```
/**
 * Buscar los pagos asociados a un comercio
 * @param idComercio
 * @return
 */
public ArrayList<PagoBean> getPagos(@WebParam(name = "idComercio")String 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.

El comando es: `wsimport -d build/client/WEB-INF/classes -p ssii2.visa http://10.5.9.1:8080/P1-ws-ws/VisaDAOWSService?wsdl`



Ahora vemos como la carpeta se ha rellenado con archivos `.class`, que son los archivos que resultan de compilar archivos `.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}`

Añadimos las siguientes líneas en `generar-stubs`:

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

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.

Realizamos los siguientes comandos para compilar, empaquetar y desplegar el cliente:

```
eps@labvirtips:~/Escritorio/P1-ws$ ant compilar-cliente
Buildfile: /home/eps/Escritorio/P1-ws/build.xml

montar-jerarquia:

compilar-cliente:
[javac] Compiling 13 source files to /home/eps/Escritorio/P1-ws/build/client/WEB-INF/classes

BUILD SUCCESSFUL
Total time: 2 seconds
eps@labvirtips:~/Escritorio/P1-ws$ ant empaquetar-cliente
Buildfile: /home/eps/Escritorio/P1-ws/build.xml

preparar-web-inf-cliente:
[copy] Copying 11 files to /home/eps/Escritorio/P1-ws/build/client

empaquetar-cliente:
[jar] Building jar: /home/eps/Escritorio/P1-ws/dist/client/P1-ws-cliente.war

BUILD SUCCESSFUL
Total time: 1 second
eps@labvirtips:~/Escritorio/P1-ws$ ant desplegar-cliente
Buildfile: /home/eps/Escritorio/P1-ws/build.xml

desplegar-cliente:
[exec] Application deployed with name P1-ws.
[exec] Command deploy executed successfully.

BUILD SUCCESSFUL
Total time: 34 seconds
eps@labvirtips:~/Escritorio/P1-ws$
```

Ejecutamos el cliente y probamos a realizar un pago:

The image shows two screenshots of a web application titled 'Sistema de Pago con tarjeta'.

The first screenshot shows the 'Proceso de un pago' (Payment Process) form. It contains the following fields and values:

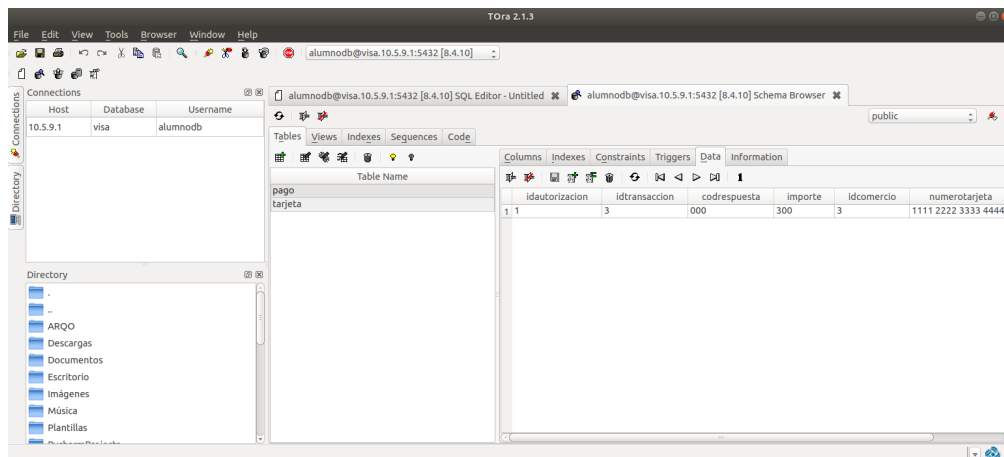
- Id Transacción: 3
- Id Comercio: 3
- Importe: 300
- Numero de visa: 1111 2222 3333 4444
- Titular: Jose Garcia
- Fecha Emisión: 11/09
- Fecha Caducidad: 11/22
- CVV2: 123
- Modo debug: ☐ True ☐ False
- Direct Connection: ☐ True ☐ False
- Use Prepared: ☐ True ☐ False
- A 'Pagar' button is at the bottom.

The second screenshot shows the 'Pago con tarjeta' confirmation page. It displays the message: 'Pago realizado con éxito. A continuación se muestra el comprobante del mismo:'. Below this, the following data is shown:

- idTransaccion: 3
- idComercio: 3
- importe: 300.0
- codRespuesta:
- idAutorizacion:

A link 'Volver al comercio' is provided at the bottom of the confirmation area.

Comprobamos que el pago se ha realizado con Tora:



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.

Cuestión 2. De los diferentes servlets que se usan en la aplicación, ¿podría indicar cuáles son los encargados de solicitar la información sobre el pago con tarjeta cuando se usa pago.html para realizar el pago, y cuáles son los encargados de procesarla?

Cuestión 3. Cuando se accede a pago.html para hacer el pago, ¿qué información solicita cada servlet? Respecto a la información que manejan, ¿cómo la comparten? ¿dónde se almacena?

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

Cuestión 1: La ruta de páginas, jsp y servlets que se visita es la siguiente:

pago.html -> ComienzaPago.java -> formdatosvisa.jsp -> ProcesaPago.java -> formdatosvisa.jsp -> muestraerror

Al haber un error al procesar el pago, se vuelve a formdatosvisa.jsp y se enseña el error producido.

Cuestión 2: El servlet encargado de solicitar la información del pago es ComiendaPago.java y el que la procesa es ProcesaPago.java

Cuestión 3: La información que se solicita es el titular de la tarjeta, el número, la fecha de emisión y de caducidad y el cvv. La información se guarda en PagoBean.java y se utiliza en formdatosvisa.jsp y ProcesaPago.java

Cuestión 4: La diferencia principal es que cuando se accede a la aplicación desde pago.html, se piden los datos de la transacción “en dos partes”, mientras que cuando se usa testdb.jsp los datos se introducen todos a la vez. Ambas opciones funcionan ya que ambos envían los datos a ProcesaPago.java por lo que, en el fondo, las dos opciones tienen la misma funcionalidad.