		Escuela Politécnica Superior Ingeniería Informática Prácticas de Sistemas Informáticos 2			
Grupo	2321	Práctica	1A	Fecha	08/03/2021
Alumno/a		Arribas Gozalo, Francisco Javier			
Alumno/a		Sáenz Ferrero, Santos			

Práctica 1A: Arquitectura de JAVA EE (Primera Parte)

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.

La IP de nuestra máquina virtual es **10.7.8.Z** (Grupo 2321, Pareja 8)

En **P1-base/build.properties** hay que cambiar:

- `as.host=localhost` → `as.host=10.7.8.1`

En **P1-base/postgresql.properties** hay que cambiar:

- `db.host=127.0.0.1` → `db.host=10.7.8.1`
- `db.client.host=127.0.0.1` → `db.client.host=10.7.8.1`

En **P1-base**, ejecutamos el comando **ant todo**.

```

root@javier-pop-os: /home/nawtilus/Documents/SI_II/P1A/P1-base
[+]

root@javier-pop-os: /home/n... x  nawtilus@javier-pop-os: ~/Do... x  nawtilus@javier-pop-os: ~/Do... x  ▼
r redeploy the application. Or if this is a new deployment, pick a different nam
e. Please see server.log for more details.
[exec] Result: 1

BUILD SUCCESSFUL
Total time: 3 seconds
root@javier-pop-os:/home/nawtilus/Documents/SI_II/P1A/P1-base# ant redesplegar
Buildfile: /home/nawtilus/Documents/SI_II/P1A/P1-base/build.xml

redesplegar:
replegar:
[exec] Command undeploy executed successfully.

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

BUILD SUCCESSFUL
Total time: 1 second
root@javier-pop-os:/home/nawtilus/Documents/SI_II/P1A/P1-base#

```

Podemos acceder a la base de datos, aquí hay una captura de la tabla “tarjeta”:

P1: tarjeta x

Table: tarjeta

P1/visa/public/TABLE/tarjeta

Info Columns Data Row Count Primary Key Indexes Grants Row Id References

In DbVisualizer Pro this feature includes data editing functionality and management of binary/BLOB data.

*	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	7557 9649 3955 5976	Irene Avila Morales	01/10	03/22	185
10	7109 3591 3164 3418	Armando Avila Pomares	11/09	08/22	041
11	0847 5890 5708 6399	Emiliano San Martin Poza	10/10	08/22	492

Realice un pago contra la aplicación web empleando el navegador en la ruta <http://10.X.Y.Z:8080/P1>

Introducimos los datos del pago:

← → ↻ 🏠 🔒 https://10.7.8.1:8181/P1/

Id Transacción:

Id Comercio:

Importe:

← → ↻ 🏠 🔒 https://10.7.8.1:8181/P1/procesapago

Pago con tarjeta

Numero de visa:

Numero de tarjeta no valido

Titular:

Fecha Emisión:

Fecha Caducidad:

CVV2:

Id Transacción: 1

Id Comercion: 1

Importe: 10.0

Prácticas de Sistemas Informáticos II

Confirmación del pago desde la página web:

← → ↻ 🏠 🔒 https://10.7.8.1:8181/P1/procesapago

Pago con tarjeta

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

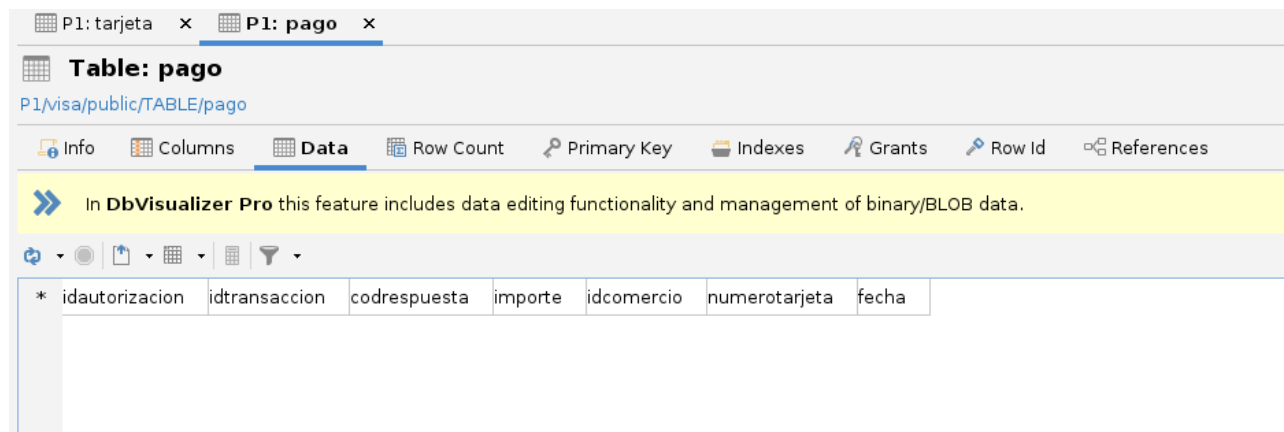
idTransaccion: 1
idComercio: 1
importe: 10.0
codRespuesta: 000
idAutorizacion: 1

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

Conéctese a la base de datos (usando el cliente Tora por ejemplo) y obtenga evidencias de que el pago se ha realizado.

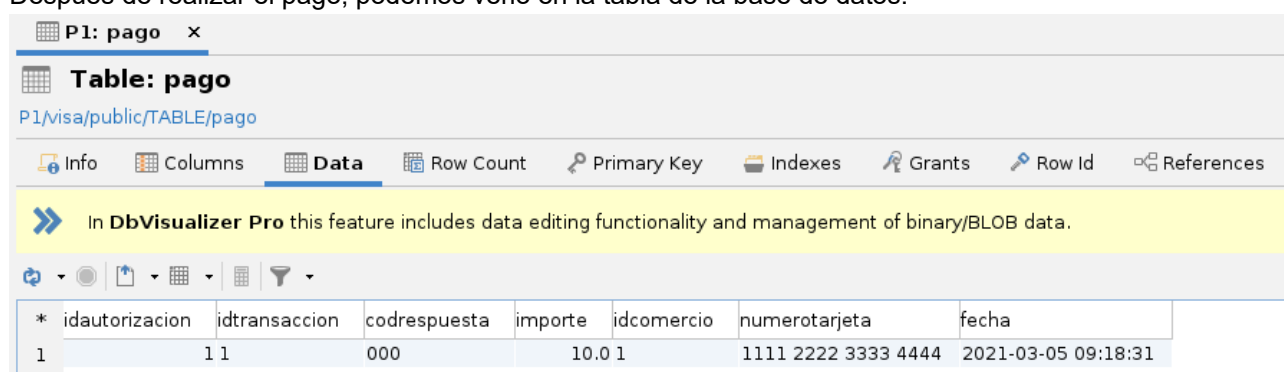
Antes de realizar el pago desde la página, hicimos una captura de la base de datos de pagos vacía:



The screenshot shows the DbVisualizer Pro interface with the 'pago' table selected. The table structure is as follows:

* idautorizacion	idtransaccion	codrespuesta	importe	idcomercio	numerotarjeta	fecha
------------------	---------------	--------------	---------	------------	---------------	-------

Después de realizar el pago, podemos verlo en la tabla de la base de datos:

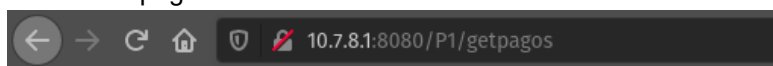


The screenshot shows the DbVisualizer Pro interface with the 'pago' table selected. The table now contains one record:

* idautorizacion	idtransaccion	codrespuesta	importe	idcomercio	numerotarjeta	fecha
1	1	000	10.0	1	1111 2222 3333 4444	2021-03-05 09:18:31

Acceda a la página de pruebas extendida, <http://10.X.Y.Z:8080/P1/testbd.jsp>. Compruebe que la funcionalidad de listado y de borrado de pagos funciona correctamente. Elimine el pago anterior.

Listado de pagos:



Pago con tarjeta

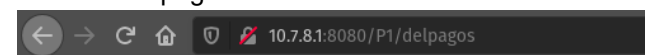
Lista de pagos del comercio 1

idTransaccion	Importe	codRespuesta	idAutorizacion
1	10.0	000	1

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

Borramos el pago realizado:



Pago con tarjeta

Se han borrado 1 pagos correctamente para el comercio 1

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

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.

El primer paso es implementar una conexión directa. Para ello, hay que modificar:

En **P1-base/src/ssii2/visa/dao/DBTester.java** hay que cambiar:

- `private static final String JDBC_DRIVER = "org.apache.derby.jdbc.ClientDriver";` →
`private static final String JDBC_DRIVER = "org.postgresql.Driver";`
- `private static final String JDBC_CONNSTRING = "jdbc:derby://10.1.1.1:1527/visa;create=true";` →
`private static final String JDBC_CONNSTRING = "jdbc:postgresql://10.7.8.1:5432/visa;create=true";`
- `private static final String JDBC_USER = "APP";` →
`private static final String JDBC_USER = "alumnodb";`
- `private static final String JDBC_PASSWORD = "APP";` →
`private static final String JDBC_PASSWORD = "*****";`

Después de esto, será necesario hacer *reboot* del *host* y volver a desplegar la aplicación.

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.

Insertar datos:

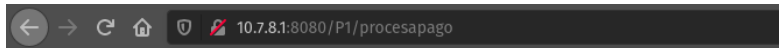


Pago con tarjeta

Proceso de un pago

Id Transacción:	<input type="text" value="2"/>
Id Comercio:	<input type="text" value="1"/>
Importe:	<input type="text" value="20"/>
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 type="radio"/> True <input type="radio"/> False
Direct Connection:	<input checked="" type="radio"/> True <input type="radio"/> False
Use Prepared:	<input type="radio"/> True <input type="radio"/> False
<input type="button" value="Pagar"/>	

Éxito:



Pago con tarjeta

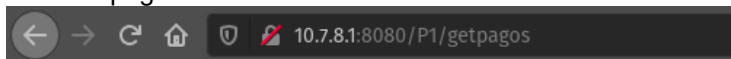
Pago realizado con éxito. A continuación se muestra el comprobante del mismo:

idTransaccion: 2
idComercio: 1
importe: 20.0
codRespuesta: 000
idAutorizacion: 1

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

Lista de pagos:



Pago con tarjeta

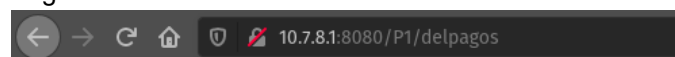
Lista de pagos del comercio 1

idTransaccion	Importe	codRespuesta	idAutorizacion
2	20.0	000	1

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

Pago borrado:



Pago con tarjeta

Se han borrado 1 pagos correctamente para el comercio 1

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

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.

En *P1-base/postgresql.properties* vemos que:

- **DataSource:** `db.datasource=org.postgresql.ds.PGConnectionPoolDataSource`
- **Nombre del *pool*:** `db.pool.name=VisaPool`

En la consola de administración comprobamos que los recursos han sido creados correctamente:

Edit JDBC Connection Pool

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

General Settings

Pool Name:	VisaPool
Resource Type:	<input type="text" value="javax.sql.ConnectionPoolDataSource"/>
	<small>Must be specified if the datasource class implements more than 1 of the interface.</small>
Datasource Classname:	<input type="text" value="org.postgresql.ds.PGConnectionPoolDataSource"/>
	<small>Vendor-specific classname that implements the DataSource and/or XADataSource APIs</small>

Valores de las variables:

Pool Settings

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

Impacto de las variables:

- **Initial and Minimum Pool Size:**
 - Es el número mínimo de *threads* concurrentes que el servidor administra. Cuando el servidor se inicializa, arrancará con este número de hilos. Por tanto, si el número es demasiado alto el servidor puede tardar demasiado en arrancar, pero si el número es demasiado bajo no se podrá dar suficiente servicio con varias conexiones preparadas para el cliente.
- **Maximum Pool Size:**
 - Número máximo de conexiones que la aplicación puede crear para satisfacer las peticiones de clientes. Si fuese demasiado bajo, el servidor no admitiría mucha concurrencia y los clientes se quedarían esperando por los servicios más tiempo, y si es demasiado alto, la máquina del servidor se puede sobrecargar y ver su rendimiento deteriorado.
- **Pool Resize Quantity:**
 - Si se cumple el *timeout* de una de las conexiones no ocupadas, éste es el número de conexiones sobre *Minimum Pool Size* que se eliminan.

- **Idle Timeout:**
 - Tiempo que una conexión libre se deja antes de declarar un *timeout* y eliminar la conexión. Este valor es importante – si es muy bajo puede causar errores de conexión, pero si es demasiado alto es un gasto de recursos innecesarios.
- **Max Wait Time:**
 - Es el tiempo máximo que espera un cliente hasta enviar una señal de *timeout* al servidor. Si el tiempo es muy bajo puede dar lugar a errores donde el servidor tenga que rehacer peticiones ya procesadas, pero si es muy alto puede llegar a no ser útil, ya que el servidor ya habría mandado un mensaje de *timeout* por sí mismo.

Ejercicio 4:

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

En P1-base/src/src/ssii2/visa/dao/VisaDAO.java:

Consulta de si una tarjeta es válida:

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

Ejecución del pago:

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

En SELECT_TARJETA_QRY se comprueba que los datos enviados por HTTP sean correctos.

En INSERT_PAGOS_QRY introduce los datos de id, importe, comercio y número de tarjeta en la tabla de pagos.

Ejercicio 5:

Edite el fichero VisaDAO.java y localice el método errorLog.

Método errorLog:

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

Compruebe en qué partes del código se escribe en *log* utilizando dicho método.

En compruebaTarjeta(TarjetaBean tarjeta):

- Si isPrepared() == true, errorLog(select);
- En caso contrario, errorLog(qry);

En realizaPago(PagoBean pago):

- Si isPrepared() == true, errorLog(insert);
- En caso contrario, errorLog(insert);

- Si `ret == true`, `errorLog(select);`
- En caso contrario, `errorLog(select);`

```
En PagoBean[] getPagos(String idComercio):
```

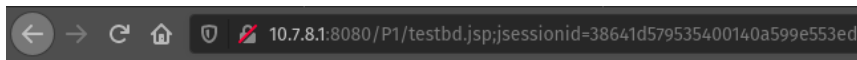
- `errorLog(qry + "[idComercio=" + idComercio + "]);`

```
En delPagos(String idComercio):
```

- `errorLog(qry + "[idComercio=" + idComercio + "]);`

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`.

Insertar datos:

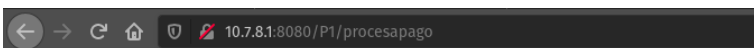


Pago con tarjeta

Proceso de un pago

Id Transacción:	<input type="text" value="5"/>
Id Comercio:	<input type="text" value="1"/>
Importe:	<input type="text" value="50"/>
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"/>	

Éxito:



Pago con tarjeta

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

```
idTransaccion: 5
idComercio: 1
importe: 50.0
codRespuesta: 000
idAutorizacion: 2
```

[Volver al comercio](#)

Incluya en la memoria una captura de pantalla del log del servidor.

Instance: **server**
Log File: **server.log**

Log Viewer Results (40)		
Records before 711	Log File Record Numbers 711 through 750	Records after 750
Record Number	Log Level	Message
750	INFO	WebModule[null] ServletContext.log():[INFO] Acceso correcto:/procesapago(details)
749	SEVERE	PWC6117: File "null" not found(details)
748	INFO	WebModule[null] ServletContext.log():[INFO] Acceso correcto:/testbd.jsp(details)
747	SEVERE	[directConnection=false] select idAutorizacion, codRespuesta from pago where idTransaccion = '5' ... (details)
746	SEVERE	[directConnection=false] insert into pago(idTransaccion,importe,idComercio,numeroTarjeta) values ('5... (details)
745	SEVERE	[directConnection=false] select * from tarjeta where numeroTarjeta='1111 2222 3333 4444' and titular... (details)
744	INFO	WebModule[null] ServletContext.log():[INFO] Acceso correcto:/procesapago(details)
743	INFO	WebModule[null] ServletContext.log():[INFO] Acceso correcto:/testbd.jsp(details)
742	INFO	WebModule[null] ServletContext.log():[INFO] Acceso correcto:/delpagos(details)
741	INFO	WebModule[null] ServletContext.log():[INFO] Acceso correcto:/testbd.jsp(details)

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.

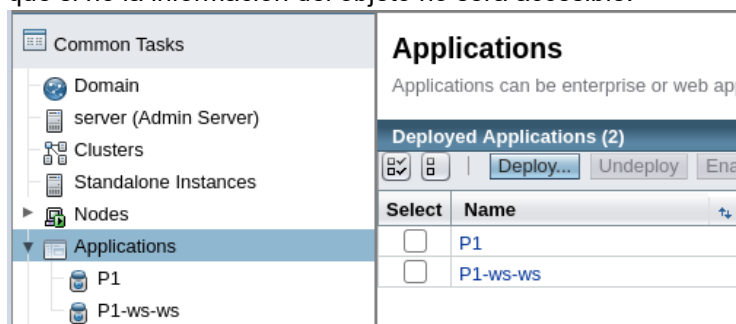
- **compruebaTarjeta()**
- **realizaPago()**
- **isDebug() / setDebug()** (Nota: VisaDAO.java contiene dos métodos setDebug que reciben distintos argumentos. Solo uno de ellos podrá ser exportado como servicio web).

Las modificaciones necesarias han sido:

- Copiar los directorios datagen, sql, web a P1-ws, copiar los ficheros postgresql.properties y postgres.xml a P1-ws, copiar el driver postgresql-jdbc-4.jar a P1-ws, cambiar build.properties para que use la IP 10.7.8.1
- Cambiar los ficheros de src/ssii2 de posición – a src/client/ssii2 o a src/server/ssii2, dependiendo del archivo
- Importar los paquetes WebMethod, WebParam y WebService de javax.jws
- Añadir @WebService a la clase VisaDAOWS.
- Añadir @WebMethod a los métodos de la clase (compruebaTarjeta, realizaPago, isPrepared, setPrepared, isDebug, uno de los métodos setDebug)
- Añadir @WebParam a los parámetros de estos métodos
- Añadir los métodos isDirectConnection y setDirectConnection, con @Override y @WebMethod, de la clase DBTester, para poder declararlos como WebMethod.
- El método realizaPago se ha cambiado de boolean a PagoBean, y por lo tanto, se ha cambiado el tipo de retorno de la función – en lugar de ret = false se devuelve ret = null y en lugar de ret = true se devuelve ret = pago

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

Para poder devolver la información sobre el pago realizado, se debe retornar el objeto entero PagoBean, ya que si no la información del objeto no será accesible.



App desplegada

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. Conteste a las siguientes preguntas:

- ¿En qué fichero están definidos los tipos de datos intercambiados con el *webservice*?
En la URL P1-ws-ws/VisaDAOWSService?xsd=1 se puede acceder a esa información
- ¿Qué tipos de datos predefinidos se usan?
xs:string, xs:int, xs:double y xs:boolean
- ¿Cuáles son los tipos de datos que se definen?
Se definen tanto clases (*tarjetaBean*) como métodos(*compruebaTarjeta*)
- ¿Qué etiqueta está asociada a los métodos invocados en el *webservice*?
Operation, que a su vez tiene las etiquetas input y output
- ¿Qué etiqueta describe los mensajes intercambiados en la invocación de los métodos del *webservice*?
Message, que a su vez tiene la etiqueta part
- ¿En qué etiqueta se especifica el protocolo de comunicación con el *webservice*?
Binding, que a su vez tiene la etiqueta soap:binding
- ¿En qué etiqueta se especifica la URL a la que se deberá conectar un cliente para acceder al *webservice*?
Soap:address

Ejercicio 8:

Realícense 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.

Primero hay que cambiar estos *import*:

```
47 // import ssii2.visa.dao.VisaDAO;  
48 import ssii2.visa.VisaDAOWSService;  
49 import ssii2.visa.VisaDAOWS;  
50 import javax.xml.ws.WebServiceRef;  
51 import javax.xml.ws.BindingProvider;  
52
```

Hay que sustituir `VisaDAO dao = new VisaDAO();` por el siguiente código:

```
// VisaDAO dao = new VisaDAO();  
VisaDAOWS dao = null;  
try {  
    VisaDAOWSService service = new VisaDAOWSService();  
    dao = service.getVisaDAOWSPort();  
} catch (Exception e) {  
    enviaError(e, request, response);  
    return;  
}
```

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.

Se debe modificar el fichero web.xml siguiendo las instrucciones del enunciado (o los comentarios del propio fichero) para añadir la URL de nuestro servidor usando la IP 10.7.8.1:

```
<context-param>
  <param-name>direccion</param-name>
  <param-value>http://10.7.8.1:8080/P1-ws-ws/VisaDAOWSService</param-value>
</context-param>
```

Es necesario volver a modificar ProcesaPago.java de la siguiente manera:

```
// VisaDAO dao = new VisaDAO();|
VisaDAOWS dao = null;
try {
  VisaDAOWSService service = new VisaDAOWSService();
  dao = service.getVisaDAOWSPort();
  BindingProvider bp = (BindingProvider) dao;
  bp.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
                             getServletContext().getInitParameter("direccion"));
} catch (Exception e) {
  enviaError(e, request, response);
  return;
}
```

Gracias a `BindingProvider` somos capaces de acceder a dirección para recibir el valor de la URL adecuado. Esta es la manera de conseguir hacer un *stub* dinámico.

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.

Debemos modificar `DelPagos.java` y `GetPagos.java` – para ello, como en el ejercicio 8, debemos añadir *import*, y como en el ejercicio 9, sustituir `VisaDAO dao = new VisaDAO();` por el código ya mostrado.

Por otro lado, en `VisaDAOWS.java` debemos cambiar los métodos `getPagos` y `delPagos` para que sean `@WebMethod`, y cambiar los parámetros de los mismos para que sean `@WebParam`.

Cambiaremos el tipo de la función `getPagos` de `PagoBean[]` a `ArrayList<PagoBean>`, y con ello modificaremos `getPagos` para que funcione la petición de pagos con el nuevo tipo de dato:

```
@WebMethod
public ArrayList<PagoBean> getPagos(@WebParam String idComercio) {
  /* Petición de los pagos para el comercio */
  // PagoBean[] pagos = dao.getPagos(idComercio);
  List<PagoBean> pagosAUX = dao.getPagos(idComercio);
  PagoBean[] pagos = pagosAUX.toArray(new PagoBean[pagosAUX.size()]);
}
```

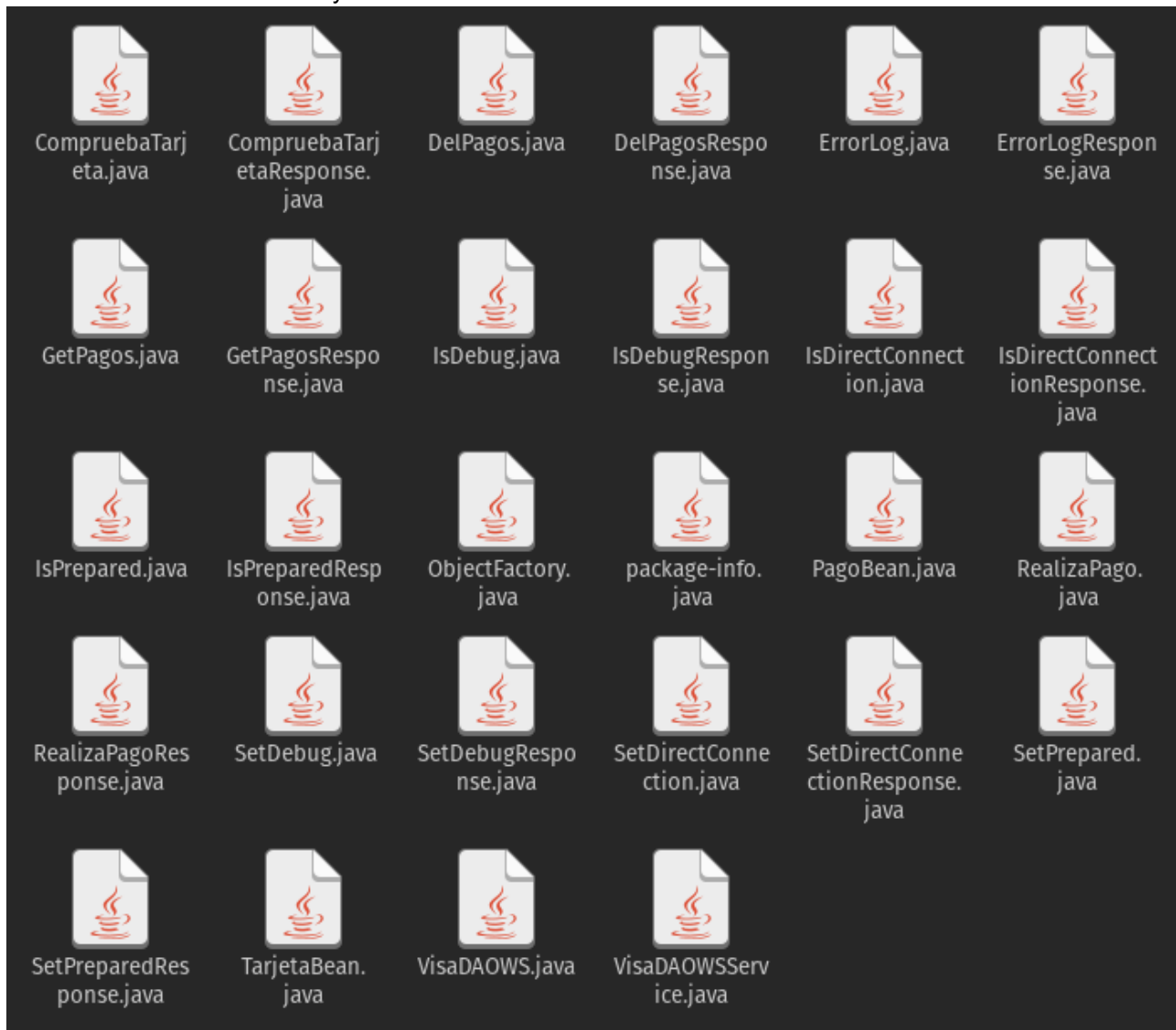
Ejercicio 11:

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 ejecutado para la importación **manual** de WSDL ha sido:

- `wsimport -d ./ -p ssii2.visa http://10.7.8.1:8080/P1-ws-ws/VisaDAOWSService?wsdl`

Este comando genera los *stubs* de cliente. Se debería poder ver que se ha generado un fichero java por cada uno de los *web services* y *web methods* del servidor.



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 `${build.client}/WEB-INF/classes`
- El paquete Java raíz(ssii2) ya está definido como `${package}`
- La URL ya está definida como `${wsdl.url}`

El comando que queremos ejecutar para la importación **dinámica** de WSDL es:

- `wsimport -d ./build/client/WEB-INF/classes -p ssii2.visa http://10.7.8.1:8080/P1-ws-ws/VisaDAOWSService?wsdl`

Para ello, se modifica el fichero XML:

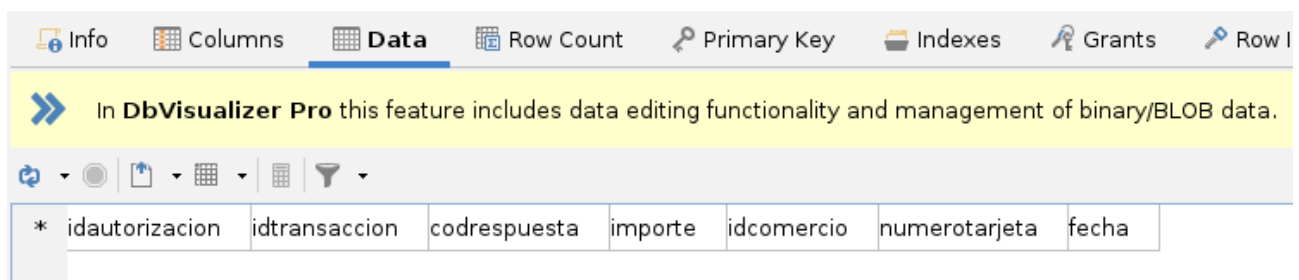
```
<target name="generar-stubs" depends="montar-jerarquia" description="Genera los stubs del cli
  <exec executable="${wsimport}">
    <arg line="-d ${build.client}/WEB-INF/classes "/>
    <arg line="-p ${paquete}.visa "/>
    <arg line="${wsdl.url}" />
  </exec>
```

Ejercicio 13:

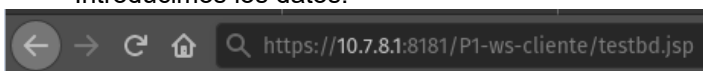
- Realice un despliegue de la aplicación completo en dos nodos tal y como se explica en la Figura 8. Habrá que tener en cuenta que ahora en el fichero build.properties hay que especificar la dirección IP del servidor de aplicaciones donde se desplegará la parte del cliente de la aplicación y la dirección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

Probamos el pago:

Base de datos vacía:



Introducimos los datos:



Éxito:

Pago con tarjeta

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

idTransaccion: 1
idComercio: 1
importe: 130.0
codRespuesta: 000
idAutorizacion: 2

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

Confirmación desde la web:

Pago con tarjeta

Lista de pagos del comercio 1

idTransaccion	Importe	codRespuesta	idAutorizacion
1	130.0	000	1

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

Pago con tarjeta

Proceso de un pago

Id Transacción:

Id Comercio:

Importe:

Numero de visa:

Titular:

Fecha Emisión:

Fecha Caducidad:

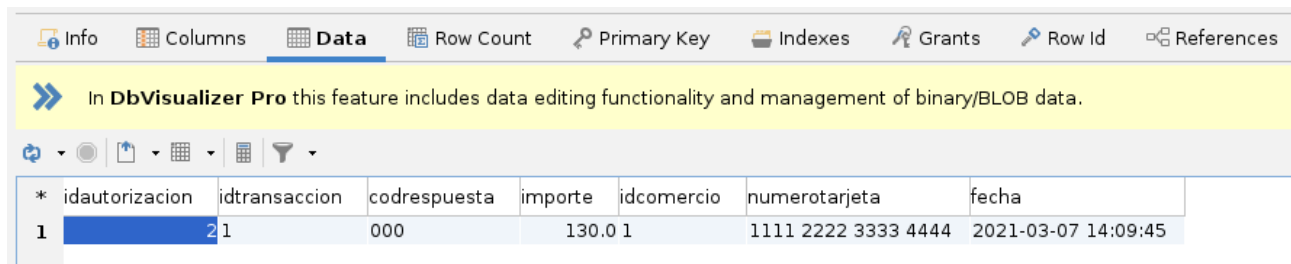
CVV2:

Modo debug: ☒ True ☐ False

Direct Connection: ☐ True ☐ False

Use Prepared: ☐ True ☐ False

Confirmación desde la base de datos:



The screenshot shows the DbVisualizer Pro interface. At the top, there are tabs for Info, Columns, Data (selected), Row Count, Primary Key, Indexes, Grants, Row Id, and References. Below the tabs, a yellow banner states: "In DbVisualizer Pro this feature includes data editing functionality and management of binary/BLOB data." Below the banner is a toolbar with icons for refresh, zoom, and other functions. The main area displays a table with the following columns: * idautorizacion, idtransaccion, codrespuesta, importe, idcomercio, numerotarjeta, and fecha. The first row of data is highlighted in blue and contains the values: 1, 21, 000, 130.0, 1, 1111 2222 3333 4444, and 2021-03-07 14:09:45.

* idautorizacion	idtransaccion	codrespuesta	importe	idcomercio	numerotarjeta	fecha
1	21	000	130.0	1	1111 2222 3333 4444	2021-03-07 14:09:45

Borramos el pago:

Pago con tarjeta

Se han borrado 1 pagos correctamente para el comercio 1

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

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.

Desde *pago.html* introducimos los datos, y el sistema accederá al servlet Comienza Pago, que enviará los datos a *formdatosvisa.jsp*, la cual tras procesar los datos enviará la información al segundo servlet Procesa Pago. Este servlet está conectado con la base de datos VisaDAO, que comprueba la correctitud de los datos. Finalmente, la query devuelve el error porque la tarjeta ha caducado, por lo que el servlet redirecciona al cliente a *error/muestraerror.jsp*, que informará al cliente del fallo en la transacción.

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?

Al usar *pago.html*, el servlet Comienza Pago solicita la información sobre el pago con tarjeta al usuario. Por otro lado, Procesa Pago es el servlet encargado de procesar dicha información, conectándose a la base de datos y aprobando/rechazando el pago.

Cuestión 3:

Quando 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?

El servlet Comienza Pago necesita un id de transacción y un id de comercio, la cantidad del importe y una url de retorno al completar el pago.

El servlet Procesa Pago necesita el número de visa, titular, fecha de emisión y caducidad de la tarjeta, y CVV2.

La información se guarda en una sesión del navegador donde ambos servlet pueden acceder. Así se comparte la información.

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?

En testbd.jsp, se accede directamente al servlet Procesa Pago, mientras que un cliente normal que acceda mediante pago.html deberá acceder al servlet Comienza Pago.

Mirando el código, podemos observar que el servlet encargado de comprobar la validez de los datos accediendo a la base de datos e inserta el pago en la misma es Procesa Pago y no Comienza Pago. Al ejecutar testbd.jsp, se accede directamente a Procesa Pago y por ello funciona correctamente.