		<b>Escuela Politécnica Superior</b> <b>Ingeniería Informática</b> <b>Prácticas de Sistemas Informáticos 2</b>			
<b>Grupo</b>	<b>2362</b>	<b>Práctica</b>	1A	<b>Fecha</b>	25/02/2020
<b>Alumno/a</b>	Ramos Pedroviejo, Alba				
<b>Alumno/a</b>	Serrano Salas, Nicolás				

## Práctica 1A: Arquitectura de Java EE

### Cuestión número 1:


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

- Modifique los ficheros que considere necesarios en el proyecto para que se despliegue tanto la aplicación web como la base de datos contra la dirección asignada a la pareja de prácticas.

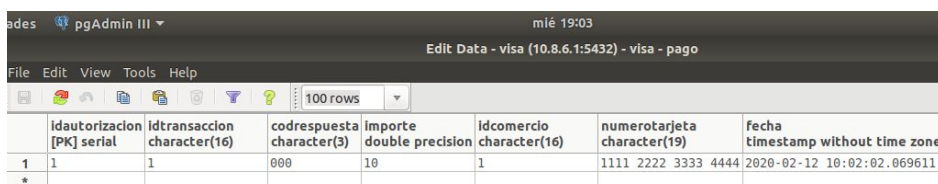
Se ha modificado la variable as.host del fichero build.properties para establecer la dirección IP 10.8.6.1 como dirección del servidor de aplicaciones. También se ha modificado el fichero postgresql.properties para establecer en la variable db.host la misma dirección, que corresponderá al servidor de base de datos y que estará en el mismo lugar. Finalmente, se ha modificado la variable db.client.host del mismo fichero para indicar que el cliente de la base de datos también estará en esa dirección.

- 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.

Si accedemos a la aplicación web y realizamos un pago con el usuario de pruebas, podemos comprobar que la aplicación responde adecuadamente, devolviendo la información correcta.




Si accedemos mediante pgAdmin a la base de datos, podremos comprobar que el pago está correctamente almacenado.



	idautorizacion [PK] serial	idtransaccion character(16)	codrespuesta character(3)	importe double precision	idcomercio character(16)	numerotarjeta character(19)	fecha timestamp without time zone
1	1	1	000	10	1	1111 2222 3333 4444	2020-02-12 10:02:02.069611

- 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.

Hemos probado a realizar un pago desde esta página usando el usuario anterior. La aplicación realiza el pago correctamente: hasta ahora llevamos dos pagos.

**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

**Pago con tarjeta**

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

idTransaccion: 2  
idComercio: 1  
importe: 5.0  
codRespuesta: 000  
idAutorizacion: 2

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

También podemos comprobar que los pagos realizados en el comercio 1 hasta ahora se muestran correctamente mediante la funcionalidad de listado:

**Pago con tarjeta**

Lista de pagos del comercio 1

IdTransaccion	Importe	codRespuesta	IdAutorizacion
1	10.0	000	1
2	5.0	000	2

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

Si probamos a borrar los pagos del comercio 1, la aplicación informa de que se han eliminado correctamente los dos pagos realizados anteriormente:

**Pago con tarjeta**

Se han borrado 2 pagos correctamente para el comercio 1

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

Se puede comprobar con pgAdmin cómo queda vacía la base de datos, por lo tanto el borrado funciona correctamente:

	idautorizacion [PK] serial	idtransaccion character(16)	codrespuesta character(3)	importe double precision	idcomercio character(16)	numerotarjeta character(19)	fecha timestamp without time zone
*							

## Cuestión número 2:

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

Adjunte en la memoria evidencias de este proceso, incluyendo capturas de pantalla.

Para realizar correctamente la conexión directa, se ha modificado el fichero DBTester.java. Concretamente, se ha establecido en la variable JDBC\_DRIVER el nombre de nuestro driver del gestor de la base de datos, que será "org.postgresql.Driver". En la variable JDBC\_CONNSTRING se ha almacenado la cadena de conexión JDBC, que será "jdbc:postgresql://host:puerto/nombreBD", siendo host nuestra IP (10.8.6.1), puerto el que se estableció en postgresql.properties (5432) y nombreBD el nombre de la base de datos (visa). Finalmente, se han modificado JDBC\_USER y JDBC\_PASSWORD para almacenar el usuario y la contraseña de la base de datos respectivamente (alumnodb, alumnodb).

Para comprobar que las modificaciones son correctas, hemos probado a realizar un pago mediante conexión directa en la página de pruebas extendidas:

**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

**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: 4

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

A continuación, hemos probado a listar los pagos del comercio 1, obteniendo la lista con el pago que acabamos de realizar. Finalmente, hemos probado el borrado y hemos observado que se ha borrado correctamente.



### Cuestión número 3:

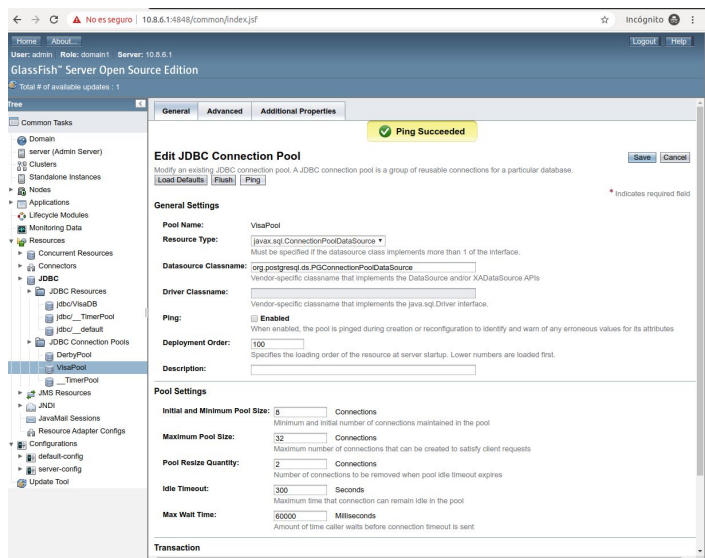
Examinar el archivo `postgresql.properties` para determinar el nombre del recurso JDBC correspondiente al `DataSource` y el nombre del pool. Acceda a la Consola de Administración. Compruebe que los recursos JDBC y pool de conexiones han sido correctamente creados. Realice un Ping JDBC a la base de datos.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

Para comprobar que los recursos que se han creado correctamente, primero miramos en el fichero `postgresql.properties` cual era el recurso y el pool que se iban a crear como veremos en la imagen a continuación de la izquierda. También accediendo a la Consola de Administración observamos en el apartado JDBC que se ha realizado correctamente el pool y en la siguiente imagen tenemos los parametros en la imagen de la derecha con los respectivos valores que se nos solicitan.

```

1 # Propiedades de la BD postgresql
2
3 # Parametros propios de postgresql
4 db.name=visa
5 db.user=alumnodb
6 db.password=****
7 db.port=5432
8 db.host=10.8.6.1
9
10 # Recursos y pools asociados
11 db.pool.name=VisaPool
12 db.jdbc.resource.name=jdbc/VisaDB
13 db.url=jdbc:postgresql://${db.host}:${db.port}/${db.name}
14 db.client.host=10.8.6.1
15 db.client.port=4848
16
17 db.delimiter=;
18 db.driver=org.postgresql.Driver

```



Estos parametros repercuten en el rendimiendo de la aplicación en el sentido de que solo puede mantener un cierto número de conexiones al mismo tiempo y se van a ir desconectando automáticamente por el servidor conexiones que han estado inactivas, dejando hueco para otras peticiones entrantes.

### Cuestión número 4:

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

- Consulta de si una tarjeta es válida.

La consulta es `getQryCompruebaTarjeta`.

## - Ejecución del pago.

La consulta es getQryInsertPago.

Incluya en la memoria de prácticas dichas consultas.

Las consultas se encuentran en el fichero VisaDAO.java:

```
85  /**
86   * getQryComprobaTarjeta
87   */
88  String getQryComprobaTarjeta(TarjetaBean tarjeta) {
89      String qry = "select * from tarjeta "
90                  + "where numeroTarjeta=" + tarjeta.getNumero()
91                  + " and titular=" + tarjeta.getTitular()
92                  + " and validaDesde=" + tarjeta.getFechaEmision()
93                  + " and validaHasta=" + tarjeta.getFechaCaducidad()
94                  + " and codigoVerificacion=" + tarjeta.getCodigoVerificacion() +
95      return qry;
96  }
97
98  /**
99   * getQryInsertPago
100  */
101  String getQryInsertPago(PagoBean pago) {
102      String qry = "insert into pago("
103                  + "idTransaccion,"
104                  + "importe,idComercio,"
105                  + "numeroTarjeta)"
106                  + " values ("
107                  + " " + pago.getIdTransaccion() + " , "
108                  + pago.getImporte() + " , "
109                  + " " + pago.getIdComercio() + " , "
110                  + " " + pago.getTarjeta().getNumero() + " "
111                  + ")";
112      return qry;
113  }
```

## Cuestión número 5:

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

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

The screenshot shows a web browser window displaying the Log Viewer application. The URL is 10.8.6.1:4848/common/logViewer/logViewer.jsf?instanceName=server&logLevel=INFO&viewResults=true#. The interface includes a search bar, a timestamp filter set to 'Most Recent', and a log level filter set to 'INFO'. Below the filters, there is a table of log entries. The table has columns for Record Number, Log Level, Message, Logger, Timestamp, and Name-Value Pairs. The log entries show a sequence of events related to a payment process, including successful accesses to testbd.jsp and insert into pago, as well as some error messages like 'File "null" not found'.

Record Number	Log Level	Message	Logger	Timestamp	Name-Value Pairs
354	INFO	WebModule[null] ServletContext.log:[INFO] Acceso correcto/procesapago(details)	javax.enterprise.web	14-feb-2020 07:54:42.069	(levelValue=800, timeMillis=1581695682069)
353	INFO	WebModule[null] ServletContext.log:[INFO] Acceso correcto/testbd.jsp(details)	javax.enterprise.web	14-feb-2020 07:54:13.494	(levelValue=800, timeMillis=1581695653494)
352	SEVERE	[directConnection=false] select idAutorizacion, codRespuesta from pago where idTransaccion = '3' ... (details)		14-feb-2020 07:53:55.366	(levelValue=1000, timeMillis=1581695635366)
351	SEVERE	[directConnection=false] insert into pago(idTransaccion,importe,idComercio,numeroTarjeta) values ('3...', (details)		14-feb-2020 07:53:55.363	(levelValue=1000, timeMillis=1581695635363)
350	SEVERE	[directConnection=false] select * from tarjeta where numeroTarjeta='1111 2222 3333 4444' and titular... (details)		14-feb-2020 07:53:55.360	(levelValue=1000, timeMillis=1581695635360)
349	INFO	WebModule[null] ServletContext.log:[INFO] Acceso correcto/procesapago(details)	javax.enterprise.web	14-feb-2020 07:53:55.335	(levelValue=800, timeMillis=1581695635335)
348	INFO	WebModule[null] ServletContext.log:[INFO] Acceso correcto/testbd.jsp(details)	javax.enterprise.web	14-feb-2020 07:53:26.783	(levelValue=800, timeMillis=1581695606783)
347	SEVERE	[directConnection=false] select idAutorizacion, codRespuesta from pago where idTransaccion = '2' ... (details)		14-feb-2020 07:49:13.530	(levelValue=1000, timeMillis=1581695335330)
346	SEVERE	[directConnection=false] insert into pago(idTransaccion,importe,idComercio,numeroTarjeta) values ('2...', (details)		14-feb-2020 07:49:13.528	(levelValue=1000, timeMillis=1581695335328)
345	SEVERE	[directConnection=false] select * from tarjeta where numeroTarjeta='1111 2222 3333 4444' and titular... (details)		14-feb-2020 07:49:13.513	(levelValue=1000, timeMillis=1581695335313)
344	INFO	WebModule[null] ServletContext.log:[INFO] Acceso correcto/procesapago(details)	javax.enterprise.web	14-feb-2020 07:49:13.505	(levelValue=800, timeMillis=1581695335305)
343	INFO	WebModule[null] ServletContext.log:[INFO] Acceso correcto/testbd.jsp(details)	javax.enterprise.web	14-feb-2020 07:48:43.525	(levelValue=800, timeMillis=1581695323525)
342	INFO	WebModule[null] ServletContext.log:[INFO] Acceso correcto/procesapago(details)	javax.enterprise.web	14-feb-2020 07:48:41.342	(levelValue=800, timeMillis=1581695321342)
341	INFO	WebModule[null] ServletContext.log:[INFO] Acceso correcto/testbd.jsp(details)	javax.enterprise.web	14-feb-2020 07:47:03.960	(levelValue=800, timeMillis=1581695223960)
340	SEVERE	PWC6117: File "null" not found(details)	org.apache.jasper.servlet.JspServlet	14-feb-2020 07:46:55.341	(levelValue=1000, timeMillis=1581695215341)
324	INFO	WebModule[null] ServletContext.log:[INFO] Acceso correcto/procesapago(details)	javax.enterprise.web	14-feb-2020 07:54:42.069	(levelValue=800, timeMillis=1581695682069)
323	INFO	WebModule[null] ServletContext.log:[INFO] Acceso correcto/testbd.jsp(details)	javax.enterprise.web	14-feb-2020 07:54:13.494	(levelValue=800, timeMillis=1581695653494)
322	SEVERE	[directConnection=false] select idAutorizacion, codRespuesta from pago where idTransaccion = '3' ... (details)		14-feb-2020 07:53:55.366	(levelValue=1000, timeMillis=1581695635366)
321	SEVERE	[directConnection=false] insert into pago(idTransaccion,importe,idComercio,numeroTarjeta) values ('3...', (details)		14-feb-2020 07:53:55.363	(levelValue=1000, timeMillis=1581695635363)



Para comprobar la diferencia entre el modo debug y no primero ejecutamos la consulta sin el modo debug, y luego con el modo debug. Se observa que aparte del log informativo de acceso correcto a /procesapago, que es común para el modo debug y no, nos aparecen 3 logs severos indicándonos las queries que se han realizado y que no han sido realizadas mediante conexión directa.

## Cuestión número 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).
- **isPrepared() / setPrepared()**

Hemos modificado el nombre de la clase VisaDAO a VisaDAOWS, y hemos modificado las referencias a la misma en todos los ficheros que la utilizaban. Además, la hemos etiquetado con el decorador @WebService(). En los métodos indicados más arriba, se ha añadido el decorador @WebMethod(operationName = "nombreMetodo") para publicar el método, y el decorador @WebParam(name = "nombreParam") para publicar cada uno de los parámetros (se añade delante de cada parámetro). Estos métodos conformarán las operaciones de nuestro servicio. Al resto de métodos que no queramos publicar (salvo el constructor de la clase) se les ha añadido el decorador @WebMethod(exclude = true).

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.

Hemos añadido estos métodos a VisaDAOWS y hemos hecho que llamen a super."método", donde método es el mismo nombre.

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

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

En esta imagen podemos observar, además, cómo y dónde se escriben los decoradores @WebMethod y @WebParam. Se haría de la misma forma para el resto de métodos que se publiquen (realizaPago, compruebaTarjeta, isDebug, setDebug, isPrepared y setPrepared).

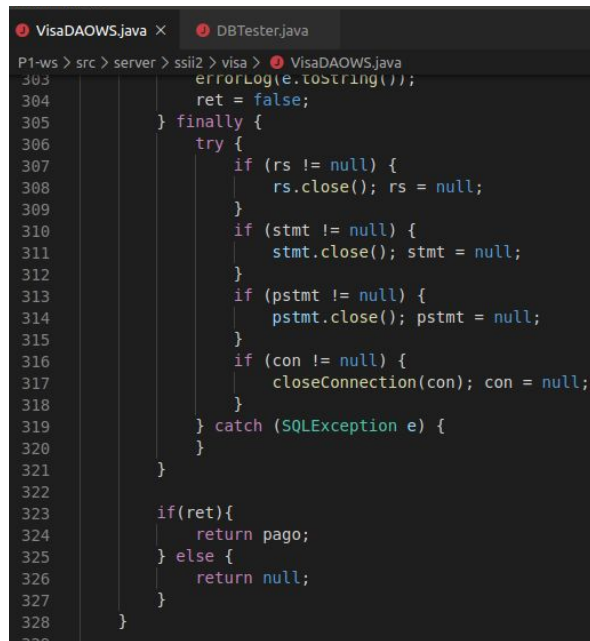
**Modifique así mismo el método realizaPago() para que éste devuelva el pago modificado tras la**

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.

Hemos cambiado el retorno de la función de boolean a PagoBean. Al final de la función, cuando se devolvía ret, hemos comprobado su valor, de forma que si es falso se devolverá un null porque algo salió mal. Si, en cambio, es true, entonces el pago ha sido correcto, contiene las modificaciones con autorización y código de respuesta y puede ser devuelto.



```
303         errorLog(e.toString());
304         ret = false;
305     } finally {
306         try {
307             if (rs != null) {
308                 rs.close(); rs = null;
309             }
310             if (stmt != null) {
311                 stmt.close(); stmt = null;
312             }
313             if (pstmt != null) {
314                 pstmt.close(); pstmt = null;
315             }
316             if (con != null) {
317                 closeConnection(con); con = null;
318             }
319         } catch (SQLException e) {
320         }
321     }
322
323     if(ret){
324         return pago;
325     } else {
326         return null;
327     }
328 }
```

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?

Porque como vamos a estar trabajando en máquinas diferentes para cliente y para servidor, y como los argumentos van a pasarse por valor y no por referencia, es necesario devolver el objeto tras su modificación en el servidor.

## Cuestión número 7:

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

Hemos utilizado las reglas compilar-servicio, empaquetar-servicio y desplegar-servicio para desplegar el servicio creado.

```
e380143@14-9-14-9:~/SI2/P1-ws$ ant compilar-servicio empaquetar-servicio desplegar-servicio
Buildfile: /home/alumnos/e380143/SI2/P1-ws/build.xml

montar-jerarquia:

compilar-servicio:
[javac] Compiling 2 source files to /home/alumnos/e380143/SI2/P1-ws/build/server/WEB-INF/classes

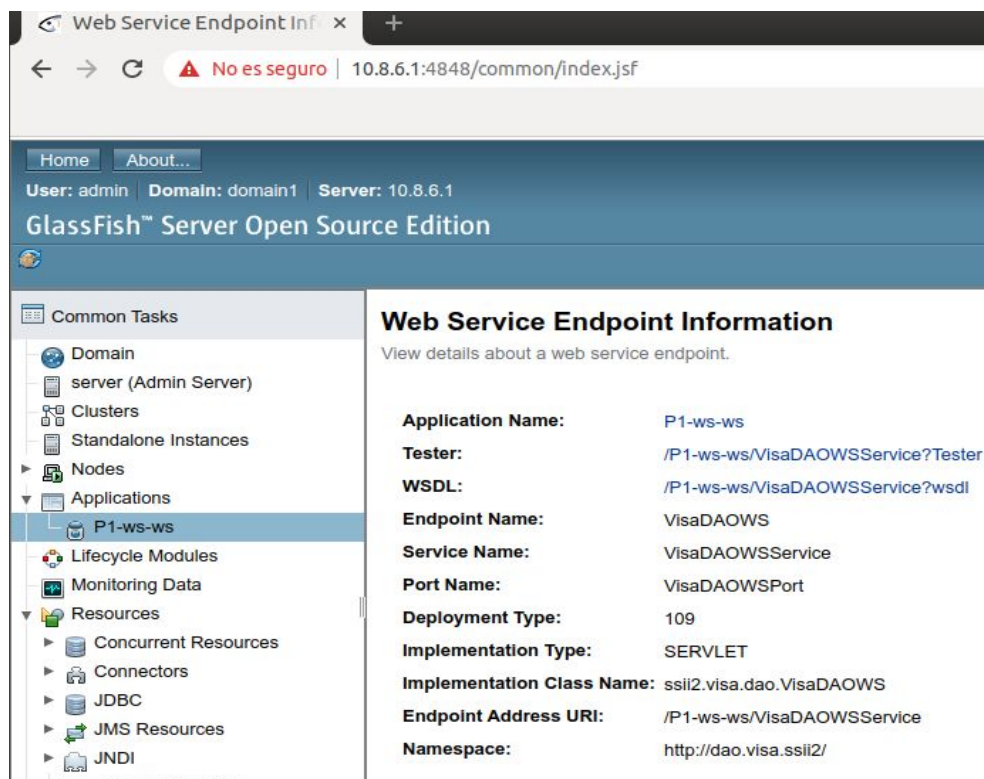
preparar-meta-inf-servicio:

empaquetar-servicio:
[delete] Deleting: /home/alumnos/e380143/SI2/P1-ws/dist/server/P1-ws-ws.war
[jar] Building jar: /home/alumnos/e380143/SI2/P1-ws/dist/server/P1-ws-ws.war

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

BUILD SUCCESSFUL
Total time: 6 seconds
```

Después hemos accedido con GlassFish a ver nuestras aplicaciones y hemos comprobado que apareciera correctamente.



The screenshot shows the GlassFish Server Open Source Edition interface. On the left, a tree view shows the hierarchy: Domain > server (Admin Server) > Clusters > Standalone Instances > Nodes > Applications > P1-ws-ws. The right pane displays the 'Web Service Endpoint Information' for the selected application. The information includes:

- Application Name:** P1-ws-ws
- Tester:** /P1-ws-ws/VisaDAOWSService?Tester
- WSDL:** /P1-ws-ws/VisaDAOWSService?wsdl
- Endpoint Name:** VisaDAOWS
- Service Name:** VisaDAOWSService
- Port Name:** VisaDAOWSPort
- Deployment Type:** 109
- Implementation Type:** SERVLET
- Implementation Class Name:** ssii2.visa.dao.VisaDAOWS
- Endpoint Address URI:** /P1-ws-ws/VisaDAOWSService
- Namespace:** http://dao.visa.ssii2/

También hemos podido acceder correctamente al fichero WSDL mediante el enlace que se observa en la imagen superior. Hemos tenido que cambiar la URL que nos dan para poner nuestra dirección IP, ya que por defecto nos ponía "si2".



The screenshot shows a web browser displaying the WSDL file for the VisaDAOWSService. The page title is 'Web Service Endpoint Information' and the URL is 'http://10.8.6.1:8080/P1-ws-ws/VisaDAOWSService?wsdl'. The XML content is displayed, showing the service definition and the endpoint address URI.

```
<?xml version='1.0' encoding='UTF-8'>
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-2004-01-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://dao.visa.ssii2/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://dao.visa.ssii2/" name="VisaDAOWSService">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://dao.visa.ssii2/" schemaLocation="http://10.8.6.1:8080/P1-ws-ws/VisaDAOWSService?xsd=1"/>
    </xsd:schema>
  </types>
  <message name="isPrepared">
    <part name="parameters" element="tns:isPrepared"/>
  </message>
  <message name="isPreparedResponse">
    <part name="parameters" element="tns:isPreparedResponse"/>
  </message>
  <message name="setPrepared">
    <part name="parameters" element="tns:setPrepared"/>
  </message>
  <message name="setPreparedResponse">
    <part name="parameters" element="tns:setPreparedResponse"/>
  </message>
</definitions>
```



**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 el fichero referenciado en la etiqueta <types>.

- **¿Qué tipos de datos predefinidos se usan? ¿Cuáles son los tipos de datos que se definen?**

En <types> podemos ver que se referencia el fichero con el esquema del servicio. Si accedemos a él, podemos ver los tipos de datos utilizados:

- Predefinidos: aquellos indicados con xs
  - xs:boolean
  - xs:string
- Los que se definen: aquellos indicados con tns:
  - tns:tarjetaBean
  - tns:pagoBean

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

En la etiqueta <messages>.

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

En la etiqueta <portType>, concretamente las etiquetas de <operation>.

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

En la etiqueta <binding>. Podemos observar el campo transport, donde se especifica que se usará el protocolo HTTP.

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

En la etiqueta <service>. Podemos observar que la URL del servicio es `http://10.8.6.1:8080/P1-ws-ws/VisaDAOWSService`.

En el fichero WSDL podemos observar que tenemos una etiqueta de <definitions>, donde podemos ver el nombre de nuestro servicio (name), el espacio de nombres (namespace), etc. También encontramos información relativa al tipo de datos que se utilizará en los mensajes en la etiqueta <types>. Podemos ver que referencia a otro fichero en el que se encontrarán los datos. En las etiquetas <message> podemos ver la definición abstracta de los datos y sus elementos (parámetros) utilizados para realizar peticiones y respuestas. En la etiqueta <portType> tenemos la representación abstracta de las operaciones y sus parámetros, es decir, los métodos de la clase VisaDAOWS que hemos publicado mediante el decorador `@WebMethod(operationName = "..")`. No incluye aquellos métodos marcados con el decorador `@WebMethod(exclude = true)`. En la etiqueta <binding> se muestra el protocolo de comunicación a utilizar. Finalmente, en <service> encontramos información sobre el servicio desplegado, concretamente la dirección IP y puerto donde se ofrece el servicio y su nombre.

## **Cuestión número 8:**

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

- El nuevo método `realizaPago()` ahora no devuelve un boolean, sino el propio objeto Pago modificado.
- Las llamadas remotas pueden generar nuevas excepciones que deberán ser tratadas en el código cliente. Incluye en la memoria una captura con dichas modificaciones

Se han añadido los imports necesarios (ssii2.visa.VisaDAOWSService, ssii2.visa.VisaDAOWS, javax.xml.ws.WebServiceRef y javax.xml.ws.BindingProvider) a la clase ProcesaPago, y se ha abierto una conexión con el servicio utilizando los stubs generados. Además, se ha protegido el código que abre la conexión con el servicio mediante un try-catch, por si ocurriese alguna excepción.

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

    BindingProvider bp = (BindingProvider) dao;
    bp.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
        getServletContext().getInitParameter("VisaDAOWSServiceURL"));
} catch (Exception e){
    e.printStackTrace();
    return;
}
```

Además, como ahora realizaPago devuelve NULL si ha salido algo mal (en vez de false) o el pago modificado si ha salido bien, hemos cambiado la comprobación que se hace cuando se la llama.

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

request.setAttribute(ComienzaPago.ATTR_PAGO, pago);
if (sesion != null) sesion.invalidate();
reenvia("/pagoexito.jsp", request, response);
return;
```

## Cuestión número 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.

Hemos definido un parámetro en web.xml que contenga la dirección del servicio:

```
<context-param>
    <param-name>VisaDAOWSServiceURL</param-name>
    <param-value>http://10.8.6.1:8080/P1-ws-ws/VisaDAOWSService</param-value>
</context-param>
```

En ProcesaPago.java, debido a tener importado javax.xml.ws.BindingProvider podremos acceder al contexto del servlet y a sus parámetros con la llamada inferior. De esta forma, evitamos tener que recompilar el código si la URL del servicio se modifica; bastaría con cambiarla en el parámetro de web.xml.

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

## Cuestión número 10:

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

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

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

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

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

Incluye en la memoria una captura con las adaptaciones realizadas

Hemos publicado el método delPagos añadiendo los decoradores adecuados:

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

Hemos modificado VisaDAOWS para publicar el método getPagos utilizando los decoradores explicados en anteriores ejercicios (imagen izquierda), modificando también el tipo de retorno y devolviendo directamente pagos, que ya es un ArrayList, en vez de transformarlo en array previamente (imagen derecha).

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

    PreparedStatement pstmt = null;
    Connection pcon = null;
    ResultSet rs = null;
    ArrayList<PagoBean> ret = null;
    ArrayList<PagoBean> pagos = null;
    String qry = null;

    try {

        // Crear una conexion u obtenerla del pool
        pcon = getConnection();
        qry = SELECT_PAGOS_QRY;
        errorLog(qry + "[idComercio=" + idComercio + "]");

        // La preparacion del statement
        // es automaticamente tomada de un pool en caso
        // de que ya haya sido preparada con anterioridad
        pstmt = pcon.prepareStatement(qry);

        pstmt.setString(1, idComercio);
        rs = pstmt.executeQuery();

        pagos = new ArrayList<PagoBean>();

        ret = pagos;

        // Cerramos / devolvemos la conexion al pool
        pcon.close();

    } catch (Exception e) {
        errorLog(e.toString());
    } finally {
        try {
            if (rs != null) {
                rs.close(); rs = null;
            }
            if (pstmt != null) {
                pstmt.close(); pstmt = null;
            }
            if (pcon != null) {
                closeConnection(pcon); pcon = null;
            }
        } catch (SQLException e) {
            errorLog(e.toString());
        }
    }

    return ret;
}
```

Además, hemos modificado GetPagos.java para realizar el mismo proceso de conexión del ejercicio

anterior, de forma que se obtenga una conexión con el servicio utilizando los stubs generados automáticamente. Finalmente, cuando llamamos a `dao.getPagos()`, obtenemos un `ArrayList`, por lo que lo transformaremos en un array mediante el método `toArray`, creando previamente el array del tamaño de la lista. De esta forma, no es necesario modificar nada de `listpagos.jsp`, pues recibirá el tipo de datos adecuado.

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    VisaDAOWSService service;
    VisaDAOWS dao;
    ArrayList<PagoBean> aux;
    PagoBean[] pagos;

    try{
        service = new VisaDAOWSService();
        dao = service.getVisaDAOWSPort();

        BindingProvider bp = (BindingProvider) dao;
        bp.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
            getServletContext().getInitParameter("VisaDAOWSServiceURL"));
    } catch (Exception e){
        e.printStackTrace();
        return;
    }

    /* Se recoge de la petición el parámetro idComercio*/
    String idComercio = request.getParameter(PARAM_ID_COMERCIO);

    /* Petición de los pagos para el comercio */

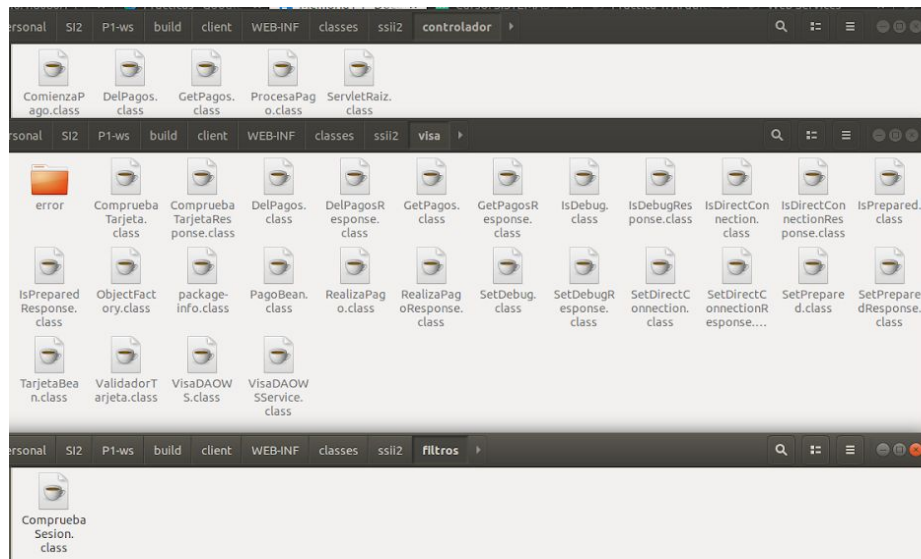
    aux = (ArrayList<PagoBean>) dao.getPagos(idComercio);
    pagos = new PagoBean[aux.size()];
    pagos = aux.toArray(pagos);

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

## Cuestión número 11:

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

Hemos utilizado el comando `wsimport -d build/client/WEB-INF/classes -p ssii2.visa http://10.8.6.1:8080/P1-ws-ws/VisaDAOWSService?wsdl` teniendo previamente desplegado el servicio. Si revisamos la carpeta, se han generado las clases de la imagen:



Estas son las clases que había en el WSDL del servicio. Mediante este comando, se copian al cliente para que disponga de ellas y pueda acceder tanto a los métodos como a los tipos de datos del servicio.

### Cuestión número 12:

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

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

Se ha completado el target como se muestra abajo, y se ha probado que funcione correctamente, obteniendo las mismas clases que en el caso anterior (previamente las hemos borrado para comprobar que se generen de nuevo).

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

### Cuestión número 13:

- Realice un despliegue de la aplicación completo en dos nodos tal y como se explica en la Figura 8.



Habr  que tener en cuenta que ahora en el fichero build.properties hay que especificar la direcci n IP del servidor de aplicaciones donde se desplegar  la parte del cliente de la aplicaci n y la direcci n IP del servidor de aplicaciones donde se desplegar  la parte del servidor. Las variables as.host.client y as.host.server deber n contener esta informaci n.

- Probar a realizar pagos correctos a trav s de la p gina testbd.jsp. Ejecutar las consultas SQL necesarias para comprobar que se realiza el pago. Anotar en la memoria pr ctica los resultados en forma de consulta SQL y resultados sobre la tabla de pagos.

La aplicaci n se ha desplegado usando dos m quinas virtuales, como se nos indic  en clase debido a problemas al utilizar dos ordenadores.

Se han modificado las variables as.host.client y as.host.server para poner las dos direcciones IP asociadas a cada uno (10.8.6.2 y 10.8.6.1, respectivamente).

Podemos comprobar que la p gina de pagos del cliente testbd.jsp se muestra correctamente y que podemos realizar un pago sin usar conexi n directa:



Podemos comprobar que la conexi n directa tambi n funciona correctamente:



Si observamos la base de datos, podemos mostrar las 100 primeras filas de la tabla y se nos muestran los pagos realizados (no coinciden con los de la imagen superior porque est n hechos en d as diferentes, pero se muestra que se almacenan correctamente):

	idautorizacion [PK] serial	idtransaccion character(16)	codrespuesta character(3)	importe double precision	idcomercio character(16)	numerotarjeta character(19)	fecha timestamp without time zone
1	1	1	000	10	1	1111 2222 3333 4444	2020-02-25 09:44:39.890082
*							

## **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.**

En primer lugar, se genera una petición de pago desde la página pago.html por parte de un comercio. El servlet ComienzaPago recibe la petición de pago y comprueba que todos los datos sean correctos. En ese caso, se muestra un formulario (llamando a formdatosvisa.jsp) para que el usuario introduzca los datos de su tarjeta. Estos datos se envían al servlet ProcesaPago, que comprobará que sean correctos. En este momento, se detectará la caducidad de la tarjeta y se mostrará un error llamando a error/muestraerror.jsp, volviendo a mostrar el formulario.

**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?**

El servlet ComienzaPago es el encargado de solicitar dicha información, pues es quien llama a formdatosvisa.jsp para que el usuario introduzca los datos de la tarjeta con la que va a pagar. Sin embargo, este servlet también se encarga de procesar que los datos introducidos sobre el pago (antes de los datos de la tarjeta) por el comercio sean válidos.

El servlet ProcesaPago es quien procesa los datos de la tarjeta para validarla y realizar el pago a través del JavaBean VisaDAO.

**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?**

El servlet ComienzaPago solicita un pago que contiene la información sobre la idTransaccion, idComercio, importe y ruta\_retonorno. ProcesaPago solicita una tarjeta que contiene el número de tarjeta, titular, fecha de emisión, fecha de caducidad y el código de verificación. Los datos de la tarjeta se almacenan en la bean también llamada TarjetaBean y los del pago en una bean llamada PagoBean. Esta información la reciben a través de HttpServletRequest y se instancian en los jsp para su uso con la etiqueta <jsp:useBean/>.

**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?**

pago.html llama al servlet ComienzaPago donde le pasa la información de la tarjeta y se validan los parámetros, luego lleva a formdatosvisa.jsp y le pasa la bean de los datos que ya tiene, donde se termina de introducir la información de la tarjeta y se validan los datos y procesa la petición llamando al servlet ProcesaPago. Sin embargo testbd.jsp como no se pasa la bean, solicita todos los parámetros para validarlos y procesar la petición.