		Escuela Politécnica Superior Ingeniería Informática Prácticas de Sistemas Informáticos 2			
Grupo	M2311	Práctica	1A	Fecha	22/02/2022
Alumno/a	Fraile, Iglesias, Javier				
Alumno/a	Fernández, París, Iván				

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

Ejercicio 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.

La primera parte de este ejercicio consiste en modificar los ficheros necesarios para poder desplegar la aplicación web, para ello tuvimos que modificar dos ficheros, "build.properties" en el que asignamos a la variables "as.host" la IP 10.4.6.3 como dirección del servidor de aplicaciones y el fichero "postgresql.properties" las variables "db.host" y "db.client.host" con la misma IP que antes para configurar la dirección de la base de datos y del cliente de la misma.

- Realice un pago contra la aplicación web empleando el navegador en la ruta <http://10.4.6.3:8080/P1> Conéctese a la base de datos (usando el cliente Tora por ejemplo) y obtenga evidencias de que el pago se ha realizado.

Accedemos a la aplicación de pruebas y realizamos una compra introduciendo los datos requeridos, podemos comprobar desde Tora realizando una consulta a la BBDD acerca de los pagos realizados, obteniendo los siguientes resultado:

Pago con tarjeta

Numero de visa:

Titular:

Fecha Emisión:

Fecha Caducidad:

Formato de fecha incorrecto

CVV2:

Id Transacción: 1
Id Comercio: 1
Importe: 9.99

Prácticas de Sistemas Informáticos II

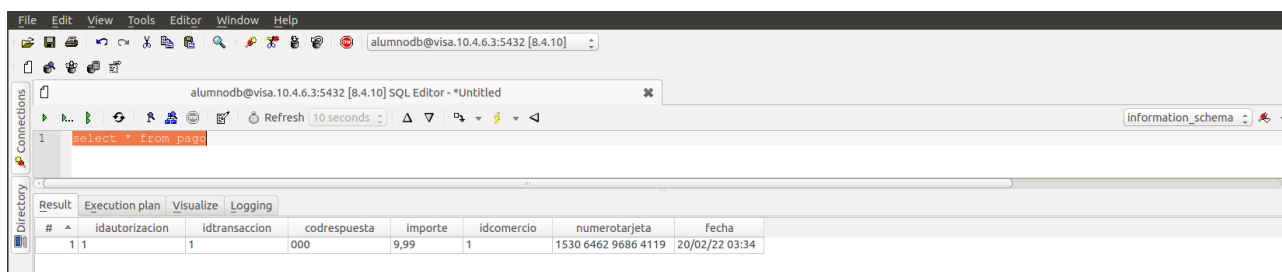
Pago con tarjeta

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

idTransaccion: 1
idComercio: 1
importe: 9.99
codRespuesta: 000
idAutorizacion: 1

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II



The screenshot shows a web browser window with the URL `alumnodb@visa.10.4.6.3:5432 [8.4.10]`. The page displays a payment confirmation message: "Pago realizado con éxito. A continuación se muestra el comprobante del mismo:". Below this, the payment details are listed: idTransaccion: 1, idComercio: 1, importe: 9.99, codRespuesta: 000, idAutorizacion: 1. There is a link "Volver al comercio".

Below the browser window, a SQL client (Tora) window is shown with the query `select * from pago` executed. The result is displayed in a table:

#	idautorizacion	idtransaccion	codrespuesta	importe	idcomercio	numerotarjeta	fecha
1	1	1	000	9.99	1	1530 6462 9686 4119	20/02/22 03:34

- Acceda a la página de pruebas extendida, <http://10.4.6.3:8080/P1/testbd.jsp>. Compruebe que la funcionalidad de listado de y borrado de pagos funciona correctamente. Elimine el pago anterior.

El pago anterior tenía como “Id comercia” el valor 1, por lo que para comprobar la funcionalidad de listado, introducimos dicho valor y pulsamos en el botón de “getPagos” y se nos muestra una tabla con ese único pago.

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

Consulta de pagos

Id Comercio:

Borrado de pagos

Id Comercio:

Prácticas de Sistemas Informáticos II

Pago con tarjeta

Lista de pagos del comercio 1

idTransaccion	Importe	codRespuesta	idAutorizacion
1	9.989999771118164	000	1

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

Ahora para comprobar que la funcionalidad de borrado de pagos funciona correctamente, indicamos cómo “idComercio”, la misma que usamos para listar anteriormente y después hacemos un listado, obteniendo una lista vacía de pagos realizados, lo que quiere decir, que el pago realizado anteriormente, ha sido eliminado de la base de datos.

Pago con tarjeta

Se han borrado 1 pagos correctamente para el comercio 1

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

Pago con tarjeta

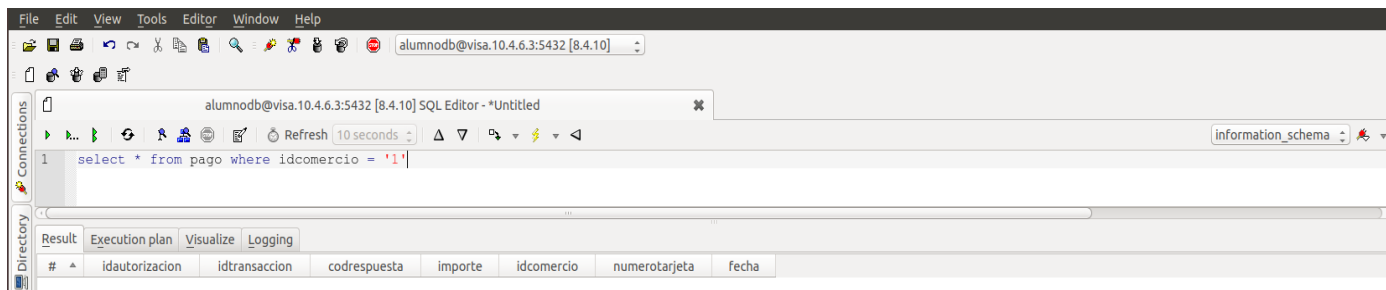
Lista de pagos del comercio 1

idTransaccion	Importe	codRespuesta	idAutorizacion
---------------	---------	--------------	----------------

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

También, si desde Tora realizamos una costa a dicho comercio, no obtenemos resultados, lo que quiere decir, que el borrado ha sido realizado correctamente.



Ejercicio 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://10.4.6.3:8080/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 poder conectarnos de forma directa debemos instanciar el *driver* JDBC del gestor de bases de datos y para obtener nuevas conexiones requerimos conocer la cadena de conexión JDBC, el usuario de la BBDD creada, es decir, visa y la contraseña de acceso a la base como se indica en el apéndice 10 del enunciado de la práctica. Para ello, accedemos al archivo "BDTester.java", en primer lugar modificamos la variable estática *JDBC_DRIVER* asignado como valor el driver de la BBDD que viene indicado en el fichero de configuración "postgresql.properties" (en *db.driver*) que es "org.postgresql.Driver". A continuación, modificamos la variable *JDBC_CONNSTRING* incluyendo la dirección IP de la BBDD que en nuestro caso es 10.4.6.3, el puerto 5432 y el nombre de la BBDD, visa, quedando entonces dicha variable de la siguiente manera "jdbc:postgresql://10.4.6.3:5432/visa". Por último, indicamos el nombre de usuario y contraseña para poder acceder a la BBDD, modificando los valores de *JDBC_USER* y *JDBC_PASSWORD* por "alumnodb" en ambos caso y terminamos haciendo un redespigüe de la app.

Una vez re-desplegada de nuevo la aplicación con los cambios comentados anteriormente, vamos a la dirección <http://10.4.6.3:8080/P1/testbd.jsp> realizamos un pago mediante conexión directa con datos diferentes a los del ejercicio 1 para evitar confusión y listamos y eliminamos dicha compra.

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="15.6"/>
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 checked="" type="radio"/> True <input type="radio"/> False
Use Prepared:	<input type="radio"/> True <input checked="" type="radio"/> False
<input type="button" value="Pagar"/>	

Pago con tarjeta

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

idTransaccion: 1
idComercio: 1
importe: 15.6
codRespuesta: 000
idAutorizacion: 1

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

Pago con tarjeta

Lista de pagos del comercio 1

idTransaccion	Importe	codRespuesta	idAutorizacion
1	15.600000381469727	000	1

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

Pago con tarjeta

Se han borrado 1 pagos correctamente para el comercio 1

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

Pago con tarjeta

Lista de pagos del comercio 1

idTransaccion	Importe	codRespuesta	idAutorizacion
---------------	---------	--------------	----------------

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

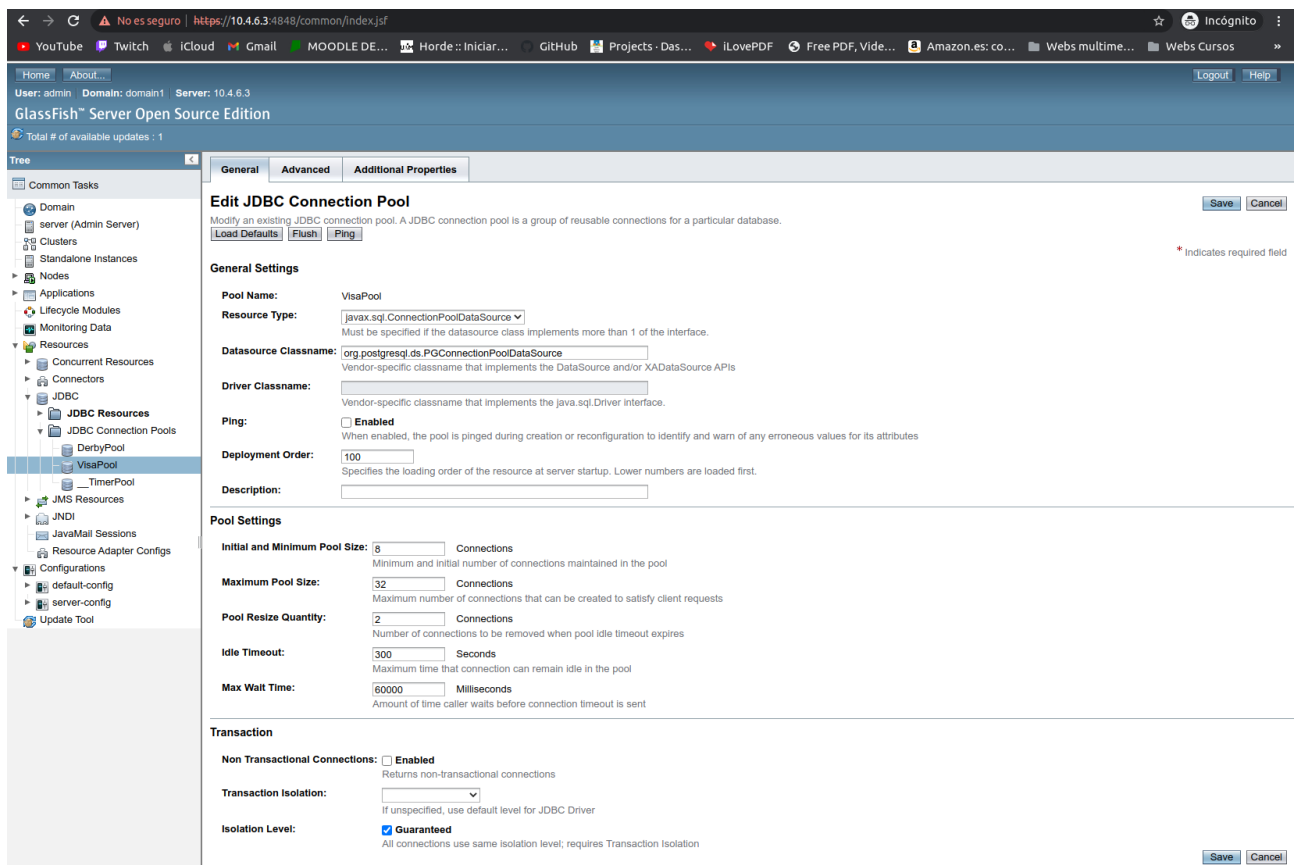
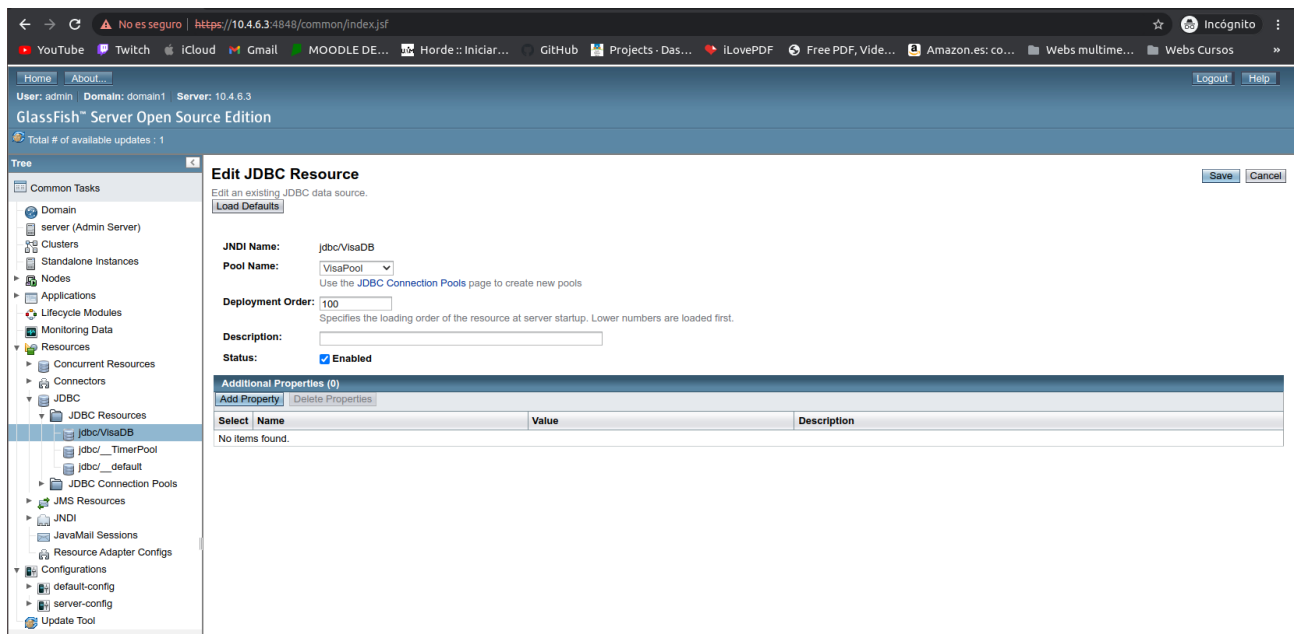
Ejercicio número 3:

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.

Primero determinamos el nombre del recurso que corresponde al `DataSource` y al nombre del pool, que vienen indicados en las variables `db.jdbc.resource.name` y `db.pool.name` respectivamente del fichero `"postgresql.properties"`

```
db.pool.name=VisaPool
db.jdbc.resource.name=jdbc/VisaDB
```

A continuación accedemos a la consola de administración, que se encuentra en la dirección <http://10.4.6.3:4848>, después al apartado de "Resources" y accedemos a la subsección "JDBC". Para poder ver el recurso del `DataSource`, accedemos a "JDBC Resources" y para el de pool, accedemos a "JDBC Connection Pool", y podemos apreciar lo siguiente.



De la imagen anterior podemos obtener los valores de: initial and Minimum Pool Size, Maximum Pool Size, Pool Resize Quantity, Idle Timeout, Max Wait Time

- Initial and Minimum Pool Size: 8 connections
- Maximum Pool Size: 32 connections
- Pool Resize Quantity: 2 connections
- Idle Timeout: 300 seconds
- Max Wait Time: 60000 milliseconds

El método de Pool de conexiones es mucho más eficiente, porque dispone de 8 a 32 conexiones a la conexión remota establecida a la base de datos, con lo cual, toda aplicación que la solicite, recibe una conexión que ya está **pre abierta** con anterioridad y lista para usar, a diferencia de la conexión directa que resulta menos eficiente porque debe abrir una por cada petición.

Cuestión número 4:

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

- Consulta si una tarjeta es válida.

La consulta asociada a comprobar si la tarjeta es válida es *getQryCompruebaTarjeta* y se encuentra en el fichero "VisaDAO.java"

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

- Ejecución del pago.

En el caso de la consulta asociada a la ejecución del pago es *getQryInsertPago* y se encuentra también en el fichero "VisaDAO.java"

```
/**
 * 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 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.

Viendo el fichero VisaDAO.java el método "errorLog()" es llamado cuando se produce una excepción para incluirla en el log y para introducir también las consultas que han sido ejecutadas que en la foto de abajo corresponden con aquellas entradas cuyo *Log Level* es *SEVERE* por lo que viendo el log del servidor se aprecia como con el modo debug se muestra más información de la que en el fichero VisaDAO.java viene indicado o ejecutado.

1. Log del servidor visto desde el portal web

Log Viewer

View, search, and filter a server log file using basic and advanced options. Refer to the Log Levels page for information about log levels you can filter here.

Advanced Search

Search Criteria

Text search:
Only log entries containing the specified text will be displayed. Search is case sensitive.

Timestamp: ☒ Most Recent
☐ Specific Range:

Log Level: ☐ Do not include more severe messages
Log entries are limited to those stored in the log file. Set appropriate log level in the Log Level page to ensure data is logged.

Modify Search

Instance:

Log File:

Log Viewer Results (40)							
Records before 147		Log File Record Numbers 147 through 186		Records after 186		T4	
Record Number	%	Log Level	%	Message	%	Logger	%
186		INFO		WebModule[null] ServletContext.log() [INFO] Acceso correcto /testbd.jsp(details)		javax.enterprise.web	
185		INFO		WebModule[null] ServletContext.log() [INFO] Acceso correcto /testbd.jsp(details)		javax.enterprise.web	
184		SEVERE		[directConnection=false] select idAutorizacion, codRespuesta from pago where idTransaccion = "1" ... (details)			
183		SEVERE		[directConnection=false] insert into pago(idTransaccion,importe,idComercio,numeroTarjeta) values (1... (details)			
182		SEVERE		[directConnection=false] select * from tarjeta where numeroTarjeta="2347 4840 5058 7931" and titular... (details)			
181		INFO		visitng: unvisited references(details)		javax.enterprise.system.tools.deployment.ddl	
180		INFO		WebModule[null] ServletContext.log() [INFO] Acceso correcto /procesapago(details)		javax.enterprise.web	
179		INFO		WebModule[null] ServletContext.log() [INFO] Acceso correcto /testbd.jsp(details)		javax.enterprise.web	
178		INFO		Admin Console: Initializing Session Attributes... (details)		org.glassfish.admingui	
177		INFO		Redirecting to index.jsp(details)		org.glassfish.admingui	
176		INFO		Listening to REST requests at context: /management/domain(details)		javax.enterprise.admin.rest	
175		SEVERE		The SSL certificate has expired [Version: V3 Subject: CN=Entrust.net Secure Server Certificate... (details)		javax.enterprise.system.security.ssl	
174		SEVERE		The SSL certificate has expired [Version: V3 Subject: CN=Entrust.net Certification Authority... (details)		javax.enterprise.system.security.ssl	
173		SEVERE		The SSL certificate has expired [Version: V3 Subject: CN=AddTrust External CA Root, OU=AddTr... (details)		javax.enterprise.system.security.ssl	
172		SEVERE		The SSL certificate has expired [Version: V3 Subject: CN=KEYNECTIS ROOT CA, OU=ROOT, O=KEYNE... (details)		javax.enterprise.system.security.ssl	
171		SEVERE		The SSL certificate has expired [Version: V3 Subject: CN=AddTrust Qualified CA Root, OU=AddTr... (details)		javax.enterprise.system.security.ssl	
170		SEVERE		The SSL certificate has expired [Version: V3 Subject: EMAILADDRESS=info@valicert.com, CN=Int... (details)		javax.enterprise.system.security.ssl	
169		SEVERE		The SSL certificate has expired [Version: V3 Subject: CN=QuoVadis Root Certification Authori... (details)		javax.enterprise.system.security.ssl	
168		SEVERE		The SSL certificate has expired [Version: V3 Subject: CN=Sonera Class1 CA, O=Sonera, C=FI ... (details)		javax.enterprise.system.security.ssl	
167		SEVERE		The SSL certificate has expired [Version: V3 Subject: CN=Thawte Timestamping CA, OU=Thawte C... (details)		javax.enterprise.system.security.ssl	
166		SEVERE		The SSL certificate has expired [Version: V3 Subject: CN=Sonera Class2 CA, O=Sonera, C=FI ... (details)		javax.enterprise.system.security.ssl	
165		SEVERE		The SSL certificate has expired [Version: V3 Subject: CN=Class 2 Primary CA, O=Certipius, C=... (details)		javax.enterprise.system.security.ssl	
164		SEVERE		The SSL certificate has expired [Version: V3 Subject: CN=GlobalSign, O=GlobalSign, OU=Globa... (details)		javax.enterprise.system.security.ssl	
163		SEVERE		The SSL certificate has expired [Version: V1 Subject: CN=GTE CyberTrust Global Root, OU="GTE... (details)		javax.enterprise.system.security.ssl	
162		SEVERE		The SSL certificate has expired [Version: V3 Subject: CN=UTN - DATACorp SGC, OU=http://www.u... (details)		javax.enterprise.system.security.ssl	
161		SEVERE		The SSL certificate has expired [Version: V3 Subject: CN=Class 3P Primary CA, O=Certipius, C=... (details)		javax.enterprise.system.security.ssl	
160		SEVERE		The SSL certificate has expired [Version: V3 Subject: CN=UTN-USERFirst Object, OU=http://www... (details)		javax.enterprise.system.security.ssl	
159		SEVERE		The SSL certificate has expired [Version: V3 Subject: CN=Deutsche Telekom Root CA 2, OU=T-Te... (details)		javax.enterprise.system.security.ssl	
158		SEVERE		The SSL certificate has expired [Version: V3 Subject: CN=Equifax Secure Global eBusiness CA... (details)		javax.enterprise.system.security.ssl	
157		SEVERE		The SSL certificate has expired [Version: V1 Subject: EMAILADDRESS=info@valicert.com, CN=Int... (details)		javax.enterprise.system.security.ssl	
156		SEVERE		The SSL certificate has expired [Version: V3 Subject: OU=Equifax Secure Certificate Authorit... (details)		javax.enterprise.system.security.ssl	
155		SEVERE		The SSL certificate has expired [Version: V3 Subject: CN=AddTrust Class 1 CA Root, OU=AddTru... (details)		javax.enterprise.system.security.ssl	
154		SEVERE		The SSL certificate has expired [Version: V3 Subject: CN=UTN-USERFirst-Hardware, OU=http://w... (details)		javax.enterprise.system.security.ssl	
153		SEVERE		The SSL certificate has expired [Version: V3 Subject: EMAILADDRESS=personal-freemail@thawte... (details)		javax.enterprise.system.security.ssl	
152		SEVERE		The SSL certificate has expired [Version: V3 Subject: CN=UTN-USERFirst-Client Authentication... (details)		javax.enterprise.system.security.ssl	
151		SEVERE		The SSL certificate has expired [Version: V3 Subject: CN=Equifax Secure eBusiness CA-1, O=Eq... (details)		javax.enterprise.system.security.ssl	
150		SEVERE		The SSL certificate has expired [Version: V3 Subject: EMAILADDRESS=server-cert@thawte.com, ... (details)		javax.enterprise.system.security.ssl	
149		SEVERE		The SSL certificate has expired [Version: V3 Subject: EMAILADDRESS=premium-server@thawte.com... (details)		javax.enterprise.system.security.ssl	
148		SEVERE		The SSL certificate has expired [Version: V3 Subject: CN=Entrust.net Secure Server Certificate... (details)		javax.enterprise.system.security.ssl	
147		SEVERE		The SSL certificate has expired [Version: V3 Subject: CN=Entrust.net Certification Authority... (details)		javax.enterprise.system.security.ssl	

Como se puede apreciar, las trazas correspondientes a la transacción son la 182 (Que hace una consulta la base de datos para comprobar que el usuario de la tarjeta existe en la base de datos), 183(Que inserta el nuevo registro a la base de datos) y 184 (Hace una consulta para mostrar en la siguiente página feedback de la transacción).

- Si desde la terminal en la que se ejecutó el comando ssh accedemos a la ruta `/opt/glassfish4.1.2/glassfish/domains/domain1/logs/` y vemos el fichero `server.log` con el comando `vi server.log` obtenemos lo siguiente:

```
ssh si2@10.4.6.3
Archivo Editar Ver Buscar Terminal Ayuda
[2022-02-20T04:26:39.587-0800] [glassfish 4.1] [SEVERE] [NCLS-SECURITY-05054] [javax.enterprise.system.security.ssl] [
tid: ThreadID=169 ThreadName=admin-listener(22)] [timeMillis: 1645359999587] [levelValue: 1000] [[
The SSL certificate has expired: [
Version: V3
Subject: CN=Entrust.net Secure Server Certification Authority, OU=(c) 1999 Entrust.net Limited, OU=www.entrust.net/C
PS incorp. by ref. (limits liab.), O=Entrust.net, C=US
Signature Algorithm: SHA1withRSA, OID = 1.2.840.113549.1.1.5

Key: Sun RSA public key, 1024 bits
modulus: 14406702391305106140142766295100576213492235373448846805360353022345240832232691736934761267893050079370845
8747092762943382762018047024013938414215872028527163969125655573945488614401571467677066488315957153441759858575776484
8486115785092156343809129655752484288535944479053857025193408242979060399539152451
public exponent: 3
Validity: [From: Tue May 25 09:09:40 PDT 1999,
To: Sat May 25 09:39:40 PDT 2019]
Issuer: CN=Entrust.net Secure Server Certification Authority, OU=(c) 1999 Entrust.net Limited, OU=www.entrust.net/CP
S incorp. by ref. (limits liab.), O=Entrust.net, C=US
SerialNumber: [ 374ad243]

Certificate Extensions: 8
[1]: ObjectID: 1.2.840.113533.7.65.0 Criticality=false
Extension unknown: DER encoded OCTET string =
0000: 04 0C 30 0A 1B 04 56 34 2E 30 03 02 04 90 ..0...V4.0....

[2]: ObjectID: 2.5.29.35 Criticality=false
AuthorityKeyIdentifier [
KeyIdentifier [
0000: F0 17 62 13 55 3D B3 FF 0A 00 6B FB 50 84 97 F3 ..b.U=...k.P...
0010: ED 62 D0 1A ..b..
]
]
1,1 Top
```

Ejercicio 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.

Las modificaciones de manera general han sido cambiar el nombre de VisaDAO.java a VisaDAOWS.java. Además aquellos métodos, parámetros y servicios públicos pero que no queremos publicar se han etiquetado con las anotaciones indicadas en el enunciado solo que métodos como "delPagos()" se le ha añadido la operación "exclude = true" para de esa forma no tener problemas de compilación y que no todos los métodos públicos queden publicados en el servicio.

```
@WebService()  
public class VisaDAOWS extends DBTester
```

- compruebaTarjeta():

```
@WebMethod(operationName = "compruebaTarjeta")  
public boolean compruebaTarjeta(@WebParam(name = "tarjeta") TarjetaBean tarjeta)
```

- realizaPago():

```
@WebMethod(operationName = "realizaPago")  
public synchronized PagoBean realizaPago(@WebParam(name = "pago") PagoBean pago)
```

- isDebug() / setDebug():

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

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


- **isDirectConnection() / setDirectConnection():**

Con la anotación “@Override” indicamos que esté método en su ejecución invoque a su método correspondiente de su clase padre y hacemos las correspondientes llamadas con super para hacer referencia a los valores del padre.

```
@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);
}
```

- **Modifique así mismo el método realizaPago() para que éste devuelva el pago modificado tras la correcta o incorrecta realización del pago:**

Los cambios fueron cambiar el tipo de retorno del método a PagoBien para de ese modo poder retornar un objeto de dicha clase, como se vio en la imagen en la que al método realizaPago se le añaden las anotaciones. Y además se añadieron las siguientes líneas de código al final del método.

```
/* Si el pago se ha realizado correctamente, devolvemos dicho
pago modificado, en caso contrario, es decir, de error, devolvemos null*/
if (ret) {
    return pago;
}

return null;
```

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

Porque ahora el archivo “VisaDAOWS.java” se encuentra en el servidor y se lo tiene que pasar ahora al cliente que se encuentra en otra máquina hay que pasarle el objeto ya modificado.

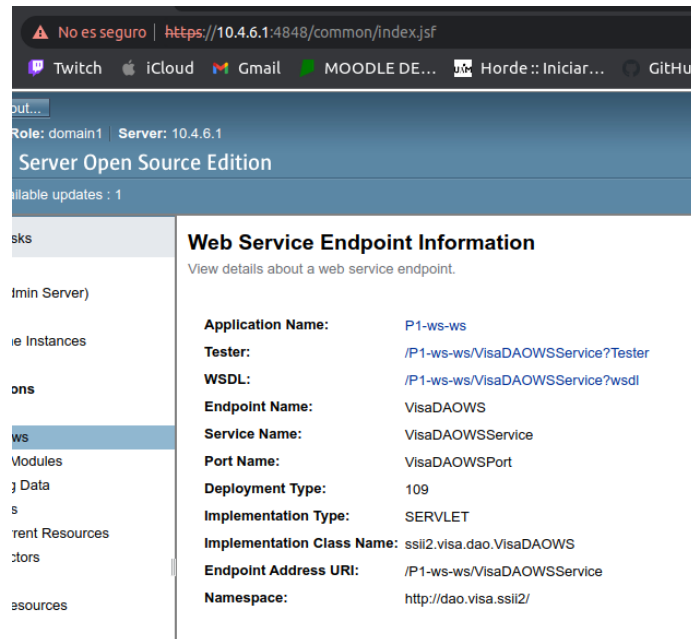
Ejercicio número 7:

Despliegue el servicio con la regla correspondiente en el build.xml. Acceda al WSDL remotamente con el navegador e inclúyelo en la memoria de la práctica.

Para desplegar, primer tuvimos que compilar el servidor mediante el comando “ant compilar-servicio”, una vez tenemos compilado el servicio, lo empaquetamos haciendo uso del comando “ant empaquetar-servicio” y por último lo desplegamos a través de “ant desplegar-servicio”. Una vez ya desplegada, nos dirigimos al panel de aplicaciones de Glassfish y observamos como la app “P1-ws” aparece.

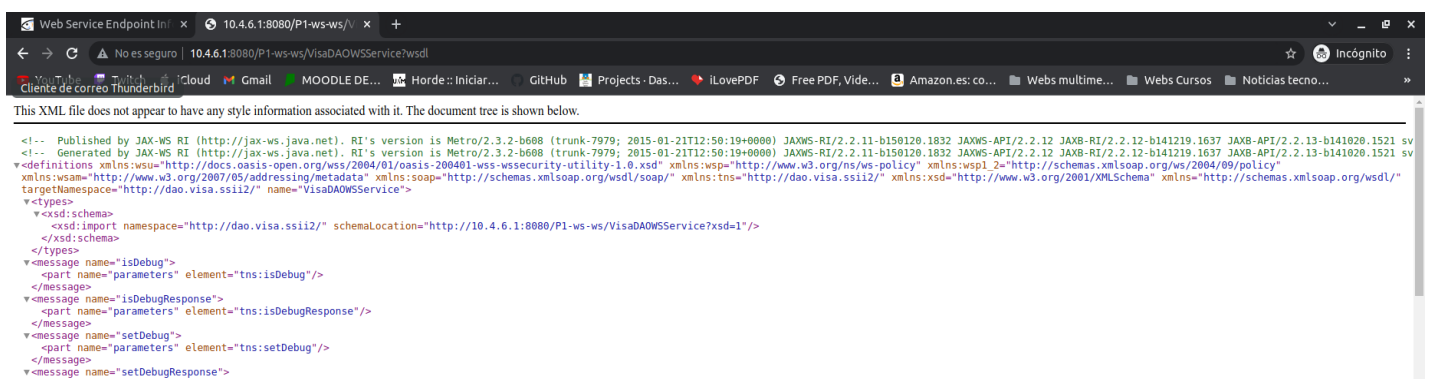
Select	Name	Deployment Order	Enabled	Engines	Action
<input type="checkbox"/>	P1	100	✓	web	Launch Redeploy Reload
<input type="checkbox"/>	P1-ws	100	✓	webservices, web	Launch Redeploy Reload

- **Pantallazo una vez pinchamos en “View Endpoint”:**



- **(habrá que asegurarse que la URL contiene la dirección IP de la máquina virtual donde se encuentra el servidor de aplicaciones):**

Comprobamos que la variable wsdl.host tiene la dirección correcta con la dirección IP del servidor, en este caso “10.4.6.1” y que podemos acceder a dicha dirección.



- **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:**

Si analizamos el fichero de arriba hacia abajo, en primer lugar podemos ver que el final de la etiqueta <definitions> se indica el nombre del servicio, es decir, “VisaDAOWSService”, a continuación, en la siguiente etiqueta llamada <types> observamos la localización donde se encuentra definido el XML schema, “schemaLocation” y el nombre de espacios o “namespace”. Después aparecen varias etiquetas de “message” Define los distintos mensajes que se intercambiarán durante el proceso de invocación del servicio y dónde vienen indicados los nombres de aquellos métodos que indicamos que queríamos publicar en nuestra app, aquellos donde se indicó la operación “exclude = true” no se ven reflejados. Debajo de la

clase “message” se encuentra el “portType” Para cada operación indica cuáles son los mensajes de entrada y salida, donde se ven aquellos métodos que quisimos publicar, de igual modo que en los “messages”. Justo debajo se encuentra la etiqueta “binding” indica el protocolo de red y el formato de los datos para las operaciones de un portType. Y al final en la etiqueta “service” viene indicado el nombre del servicio desplegado y su dirección de despliegue.

- **¿En qué fichero están definidos los tipos de datos intercambiados con el webservice?**

Los tipos de datos vienen definidos en la etiqueta <types>, se pueden importar la definición desde un fichero de esquema (xsd).

- **¿Qué tipos de datos predefinidos se usan?**

“xs:string” y “xs:boolean” son los datos predefinidos que se usan.

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

“tns:pagoBean” y “tns:tarjetaBean” son los tipos de datos que se definen.

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

Dentro de la etiqueta <portType>, se encuentran las operaciones con la etiqueta <operation>, donde se muestran todos los métodos soportados, todos ellos con un input y output referenciando a los mensajes.

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

Los mensajes intercambiados son identificados con la etiqueta <message>. Que pueden tener parámetros de entrada y/o de salida

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

Se especifica dentro de la etiqueta <binding>, la etiqueta <soap:binding> con un parámetro de “transport” donde se especifica el protocolo de transporte.

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

La URL viene indicada en la etiqueta <service>, siendo en nuestro caso <http://10.4.6.1:8080/P1-ws-ws/VisaDAOWSService>

(Información obtenida gracias a <http://www.jtech.ua.es/j2ee/publico/servc-web-2012-13/sesion01-apuntes.html>)

Ejercicio 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:

Como bien se comentó en la sección 7.7 y mirando el código, modificamos la clase ProcesaPago que es donde se llama al método realizaPago().

- **El nuevo método realizaPago() ahora no devuelve un boolean, sino el propio objeto Pago modificado.**

En la sección 5, se realizaron una serie de modificaciones en el servicio, una de ellas fue cambiar el retorno de la función realizaPago() para que devolviera null en caso de realizarse correctamente la modificación o el propio pago modificado si todo salió bien, es por ello, que el método que llama a realizaPago(), hay que modificarlo para en vez de que espere “false” para generar una excepción, la genere si el retorno es “null”.

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

- Las llamadas remotas pueden generar nuevas excepciones que deberán ser tratadas en el código cliente.

Ahora, la instanciación de la clase remota VisaDAOWS se realiza en dos pasos mediante una serie de llamadas remotas, que pueden generar excepciones, es por ello por lo que mediante un try catch podemos capturar las posibles excepciones y evitar problemas posteriores.

```
/* La instanciación de la clase remota pasa a hacerse en dos pasos */
VisaDAOWS dao = null;
VisaDAOWSService service = null;

/* Como se indica en el enunciado, en caso de una posible excepción
esta será capturada */
try {
    service = new VisaDAOWSService();
    dao = service.getVisaDAOWSPort ();
} catch (Exception e) {
    e.printStackTrace();
    return;
}
```

Los 3 imports (import ssii2.visa.VisaDAOWSService; import ssii2.visa.VisaDAOWS; import javax.xml.ws.WebServiceRef;) comentados en la sección 7.7 se incluyen en la clase ProcesaPago() del cliente.

Ejercicio número 9:

Modifique la llamada al servicio para que la ruta (URL) al servicio remoto se obtenga del fichero de configuración web.xml. Para saber cómo hacerlo consulte el apéndice 15.1 para más información y edite el fichero web.xml y analice los comentarios que allí se incluyen.

En el fichero web.xml situado en la carpeta web/WEB-INF había una parte que se encontraba comentada que indicaba que servía para la inicialización de parámetros de contexto, la descomentamos y modificamos para utilizar dicha ruta basándonos en el fichero .xml que revisamos en el ejercicio 7.

```
<?xml version="1.0" encoding="UTF-8"?>
<service name="VisaDAOWSService">
  <port name="VisaDAOWSPort" binding="tns:VisaDAOWSPortBinding">
    <soap:address location="http://10.4.6.1:8080/P1-ws-ws/VisaDAOWSService"/>
  </port>
</service>
```

Sacando el nombre del servicio y la localización o address.

```
<context-param>
  <param-name>VisaDAOWSService</param-name>
  <param-value>http://10.4.6.1:8080/P1-ws-ws/VisaDAOWSService</param-value>
</context-param>
```

```
/* Una vez instanciada la clase VisaDAOWS, obtenemos los
parámetros de inicialización */
BindingProvider bp = (BindingProvider) dao;
bp.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
getServletContext().getInitParameter("VisaDAOWSService"));
```

Ejercicio 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:

En ambas clases, los cambios son similares, añadir los imports adecuados y hacer la instanciación de la clase remota en dos pasos y capturando la posible excepción.

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

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

    /* La instanciación de la clase remota pasa a hacerse en dos pasos */
    VisaDAOWS dao = null;
    VisaDAOWSService service = null;

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

        /*
         * Una vez instanciada la clase VisaDAOWS, obtenemos los
         * parámetros de inicialización
         */
        BindingProvider bp = (BindingProvider) dao;
        bp.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
            getServletContext().getInitParameter("VisaDAOWSService"));
    } catch (Exception e) {
        e.printStackTrace();
        return;
    }
}
```

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

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

    /* La instanciación de la clase remota pasa a hacerse en dos pasos */
    VisaDAOWS dao = null;
    VisaDAOWSService service = null;

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

        /*
         * Una vez instanciada la clase VisaDAOWS, obtenemos los
         * parámetros de inicialización
         */
        BindingProvider bp = (BindingProvider) dao;
        bp.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
            getServletContext().getInitParameter("VisaDAOWSService"));
    } catch (Exception e) {
        e.printStackTrace();
        return;
    }
}
```

Además, en el caso de GetPagos(), se hace la llamada al getPagos() de VisaDAOWS y como ahora se devuelve un ArrayList hay que realizar una conversión a una lista de PagoBean (PagoBean[]). Para

ello nos basamos en cómo estaba hecho en VisaDAOWS antes del cambio.

```
/* Debemos realizar la conversión de ArrayList a PagoBean[] */
ArrayList<PagoBean> ret = (ArrayList<PagoBean>) dao.getPagos(idComercio);
PagoBean[] pagos = new PagoBean[ret.size()];
pagos = ret.toArray(pagos);
```

Tenga en cuenta que no todos los tipos de datos son compatibles con JAXB (especifica cómo 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`.

Lo primero fueron hacer públicos los métodos `getPagos()` y `delPagos()` de la clase `VisDAOWS` usando las etiquetas `@webMethod` y sus parámetros con `@webParam` y cambial el retorno del método `getPagos()` a un `ArrayList` de `PagoBean`.

```
@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;
```

```
/* Ahora ret y pagos son dos ArrayList y no hay que convertir nada */
ret = pagos;
```

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

Ejercicio 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. Tenga en cuenta que el servicio debe estar previamente desplegado.

Estando el servicio previamente desplegado y para poder ejecutar el comando, hay que saber cuál sería el directorio de salida, en este caso `build/client/WEB-INF/classes`, el paquete, que como se indica en el enunciado es `ssii2.visa` y la ruta en la que se encuentra el fichero WSDL, "<http://10.4.6.1:8080/P1-ws-ws/VisaDAOWSService?wsdl>", ruta a la que se accedió para ver el fichero .xml del que se habló en el ejercicio 7.

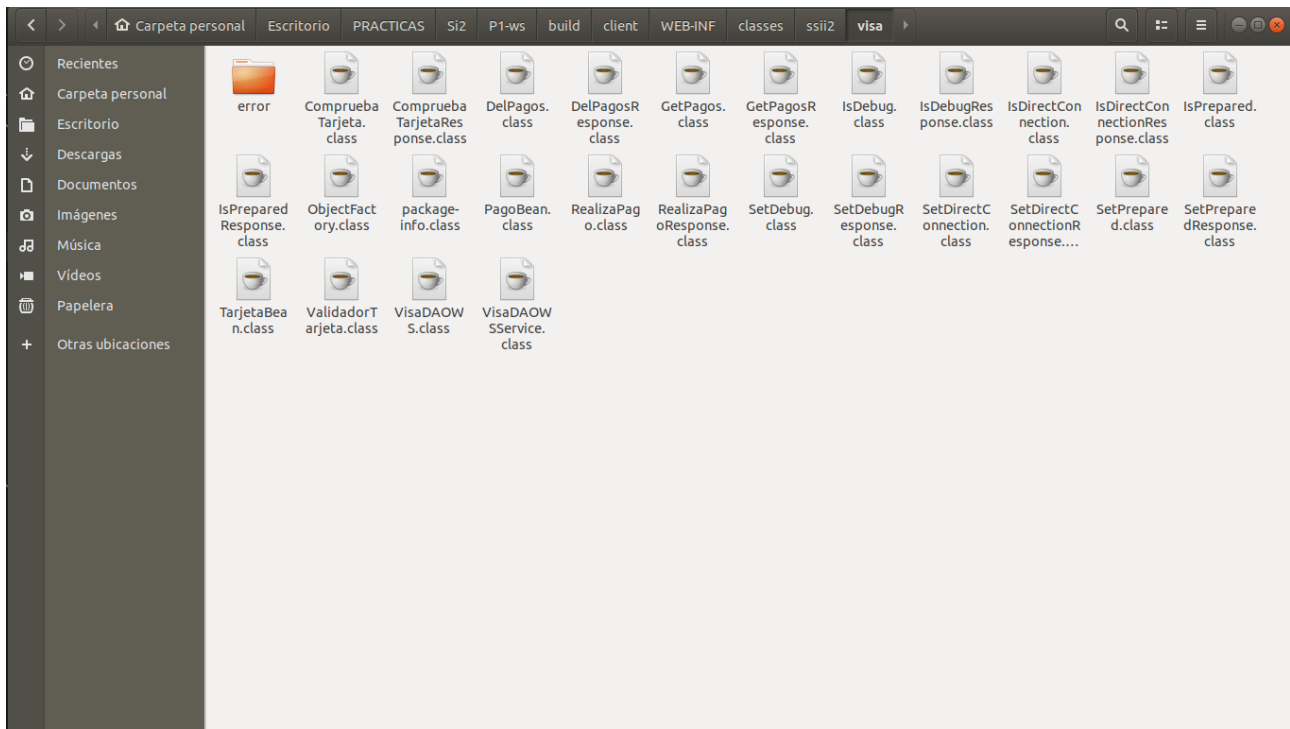
Ejecutamos el comando: `wsimport -d build/client/WEB-INF/classes -p ssii2.visa http://10.4.6.1:8080/P1-ws-ws/VisaDAOWSService?wsdl`.

```
eps@eps:~/Escritorio/PRACTICAS/Si2/P1-ws$ wsimport -d build/client/WEB-INF/class
es -p ssii2.visa http://10.4.6.1:8080/P1-ws-ws/VisaDAOWSService?wsdl
analizando WSDL...

Generando código...

Compilando código...
```


Tras ejecutar el comando, en la carpeta “classes” encontramos los siguientes archivos generados.



Las clases que se aprecian son las publicadas cuando se desplegó el servidor y que además aparecen en el fichero wsdl, de esta manera el cliente podrá hacer uso de los métodos publicados por el servidor.

Ejercicio 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}

Basándonos en cómo se ejecuta el target “desplegar-cliente” y teniendo en cuenta los 3 puntos comentados anteriormente completamos el target “generar-stubs” tal que:

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

Ejercicio número 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.

Modificamos el fichero “build.properties”, más concretamente las variables “as.host.cliente” y

“as.host.server” con las ips 10.4.6.2 y 10.4.6.1 respectivamente.

Realizamos una serie de pagos sin realizar conexión directa:



Pago con tarjeta

Numero de visa:
Titular:
Fecha Emisión:
Fecha Caducidad:
CVV2:

Id Transacción: 1
Id Comercion: 1
Importe: 13.0

Prácticas de Sistemas Informáticos II

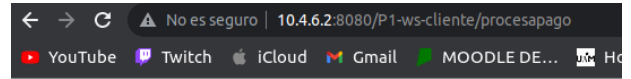


Pago con tarjeta

Numero de visa:
Titular:
Fecha Emisión:
Fecha Caducidad:
CVV2:

Id Transacción: 1
Id Comercion: 3
Importe: 17.8

Prácticas de Sistemas Informáticos II



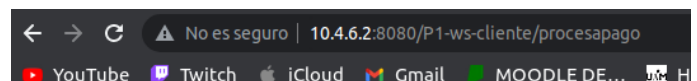
Pago con tarjeta

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

idTransaccion: 1
idComercio: 1
importe: 13.0
codRespuesta:
idAutorizacion:

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II



Pago con tarjeta

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

idTransaccion: 1
idComercio: 3
importe: 17.8
codRespuesta:
idAutorizacion:

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

alumnodb@visa.10.4.6.1:5432 [8.4.10] SQL Editor - *Untitled								
select * from pago								
Result	Execution plan	Visualize	Logging					
#	^	idautorizacion	idtransaccion	codrespuesta	importe	idcomercio	numerotarjeta	fecha
1	1		1	000	13	1	1111 2222 3333 4444	24/02/22 03:14
2	2		1	000	17,8	3	6513 0633 4651 1154	24/02/22 03:17

- **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.**

En un primer instante, nos daba fallo porque antes de implementar un cliente y un servidor por separado, usamos la conexión con la IP 10.4.6.3 y por ello nos daba problema, con cambiar en el fichero “DBTester.java” por la IP 10.4.6.1 ya pudimos realizar compras mediante conexión directa, listando después la compra realizada con la consulta de datos desde “testdb.jsp” y haciendo una consulta a la BBDD desde Tora. A continuación dejamos evidencias de ellos.

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

Consulta de pagos

Id Comercio:

Pago con tarjeta

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

idTransaccion: 1
idComercio: 4
importe: 19.7
codRespuesta:
idAutorizacion:

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

Pago con tarjeta

Lista de pagos del comercio 4

idTransaccion	Importe	codRespuesta	idAutorizacion
1	19.700000762939453	000	1

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

alumnodb@visa.10.4.6.1:5432 [8.4.10] SQL Editor - *Untitled

select * from pago

Result Execution plan Visualize Logging

#	idautorizacion	idtransaccion	codrespuesta	importe	idcomercio	numerotarjeta	fecha
1	1	1	000	19,7	4	7772 8952 5915 5042	24/02/22 03:31

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:

Tras introducir la info en el html pago.html, se realiza una petición al Servlet "Comienza Pago" que selecciona la vista que proporciona el fichero "formatosvisa.jsp", después de introducir los datos solicitados y pulsar en pagar, el jsp procesa una respuesta al Servlet "Procesa Pago" donde tras comprobar que la fecha de caducidad es correcta proporciona una respuesta al navegador web a través del jsp "muestraerror.jsp".

Cuestión 2: De los diferentes servlets (recuerde que las páginas jsp también se compilan a un servlet) que se usan en la aplicación, ¿podría indicar cuáles son los encargados de obtener la información sobre el pago con tarjeta cuando se usa pago.html para realizar el pago, y cuáles son los encargados de procesarla? ¿Qué información obtiene y procesa cada uno?

El servlet de "ComienzaPago", es el encargado de obtener la información inicial del pago, con los parámetros obtenidos de "pago.html", con datos como el id de la transacción, el id del comercio y el precio de la transacción .

Por otro lado, una vez valida estos datos, nos manda a una página dinámica llamada "formdatos.jsp", la cual es otro formulario para la introducción de los datos de la tarjeta, el titular, la fecha de emisión y la caducidad; una vez se pulsa el botón de confirmar, esta página invoca el servlet de "ProcesaPago", el cual se encarga de hacer el proceso de validación y confirmación de la transacción en la base de datos.

Cuestión 3: ¿Dónde se crea la instancia de la clase pago cuando se accede por pago.html? ¿Y cuándo se accede por testbd.jsp? Respecto a la información que manejan, ¿cómo la comparte entre los distintos servlets? ¿dónde se almacena? ¿dónde se crea ese almacén?

Cuando accedemos por "pago.html", se redirecciona al servlet de "ComienzaPago", el cual se encarga de crear una instancia de "PagoBean".

Cuando se accede por "testbd.jsp", esté directamente redirecciona a "ProcesaPago", ya que esta clase es capaz directamente de hacer los dos pasos ya que contiene una comprobación de si el pago == null, lo crea.

Los servlets como heredan de la misma clase raíz "ServletRaiz", comparten atributos HTTP, pero además, gracias a los métodos de "getSession" y "setSession", son capaces de pasar atributos de manera muy sencilla, ya que acceden a la session con ese método, y posteriormente van estableciendo u obteniendo atributos con los métodos de "setAttribute" y "getAttribute".

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?

Como ya se ha comentado en la cuestión 2, como tal la diferencia entre comenzar por "pago.html", es que primeramente llega al servlet "ComienzaPago", dónde instancia "PagoBean", y a continuación mediante el formulario "formdatosvisa.jsp", redirecciona al servlet "ProcesaPago", el cual crea la tarjeta, valida y realiza la transacción guardando en la base de datos los cambios.

Sin embargo, este proceso se puede hacer tan solo en un paso mediante el formulario completo de "testbd.jsp", ya que este redirecciona al servlet de "ProcesaPago", que en el caso de no tener una instancia de "PagoBean" (comprobando en la sesión, el atributo de sesión de pago, que en el caso de llegar de "ComienzaPago", sí que tendría un valor ya que desde ahí se hace un set del pago), la crea (comprobación if(pago == null)). Con lo cual el proceso sería el mismo.