|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | Escuela Politécnica Superior  Ingeniería Informática  Prácticas de Sistemas Informáticos 2 | | | |
| **Grupo** | **1391** | **Práctica** | **1A** | **Fecha** | 28/02/2022 |
| **Alumno/a** | | Gil Maroto, Lucía | | | |
| **Alumno/a** | | Varela Sánchez, Daniel | | | |
| **Alumno/a** | | Martín-Coello Juárez, Guillermo | | | |

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

## Ejercicio número 1:

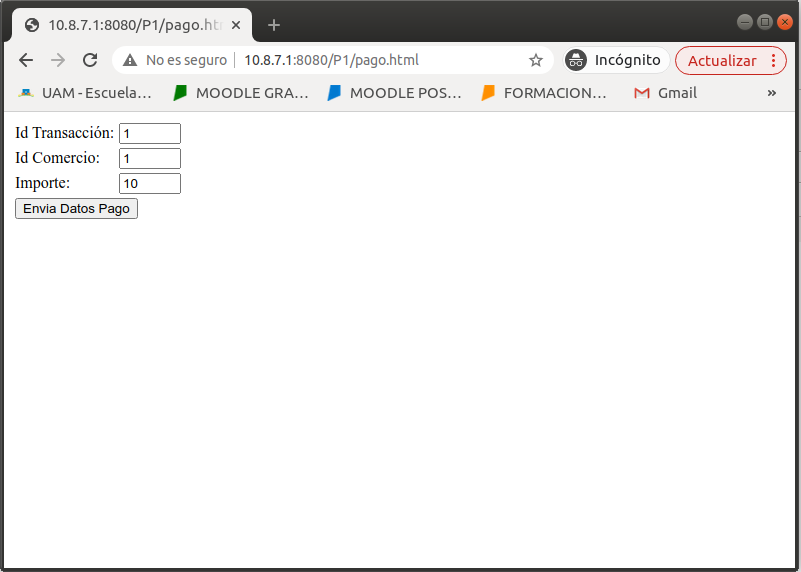
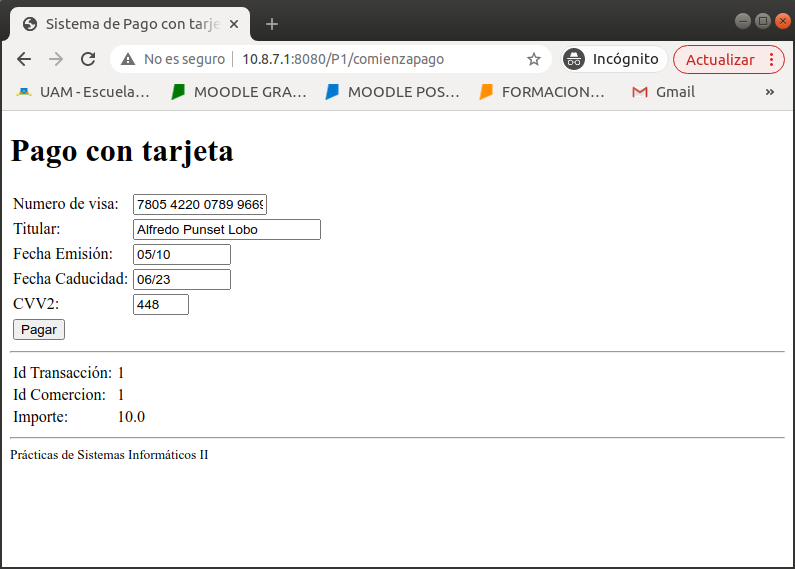
**Prepare e inicie una máquina virtual a partir de la plantilla si2srv con: 1GB de RAM asignada, 2 CPUs. A continuación:**

* **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.**

En el fichero build.properties hemos cambiado la variable as.host a 10.8.7.1, que es nuestra dirección ip del servidor de aplicaciones. También hemos modificado en el fichero postgresql.properties la variable db.host a 10.8.7.1 (que es el servidor de la base de datos) y el fichero db.client.host a 10.8.7.1 (que es el cliente de la base de datos).

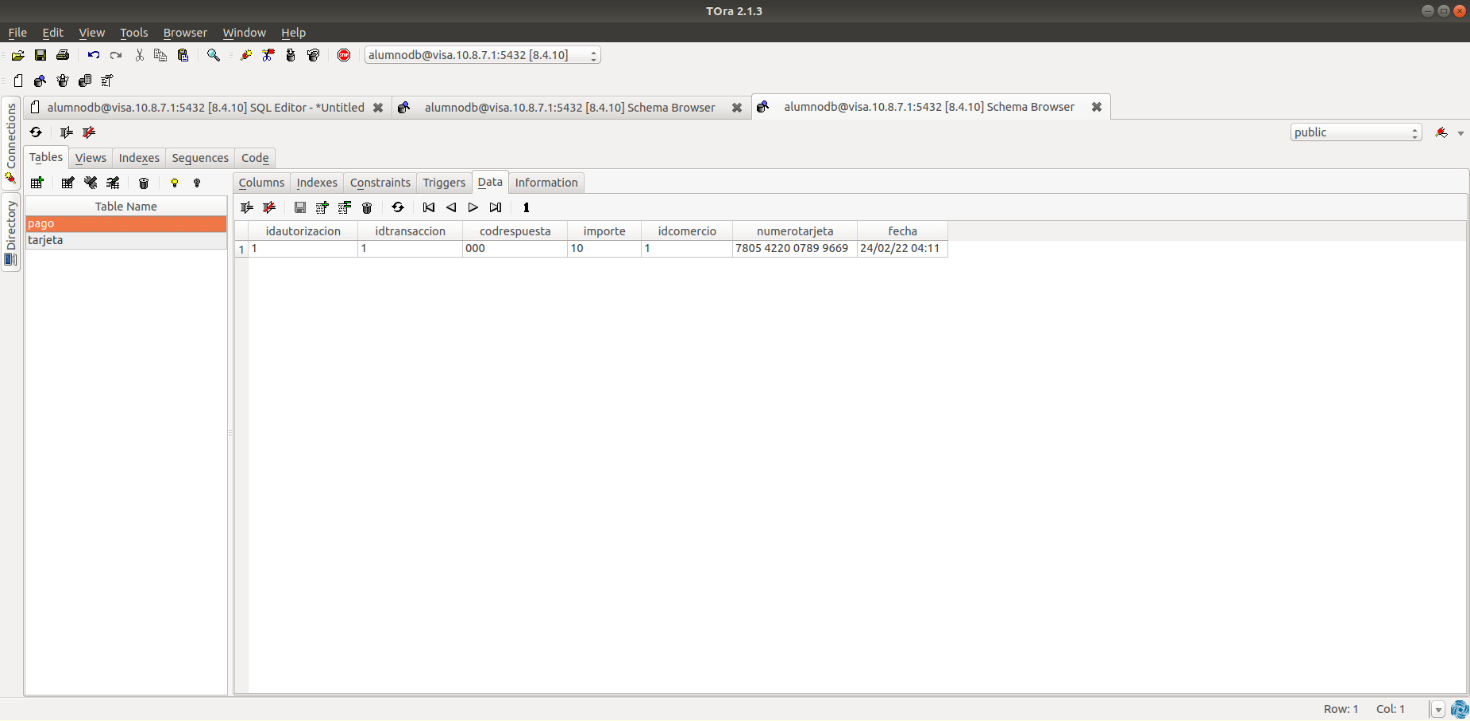
* **Realice un pago contra la aplicación web empleando el navegador en la ruta** [**http://10.X.Y.Z:8080/P1**](http://10.X.Y.Z:8080/P1)**. Conéctese a la base de datos (usando el cliente Tora por ejemplo) y obtenga evidencias de que el pago se ha realizado.**

Accedemos a la aplicación web y realizamos el pago habiendo sacado de la base de datos en TORA los datos necesarios para que el pago sea correcto.



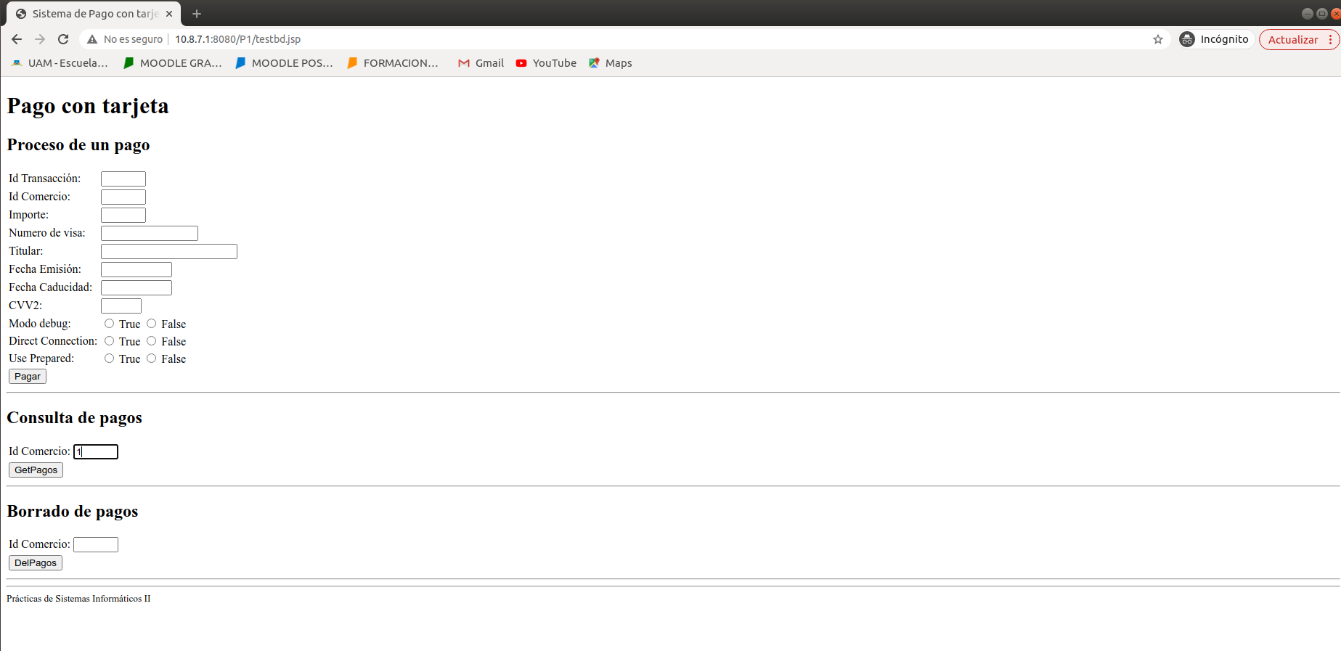
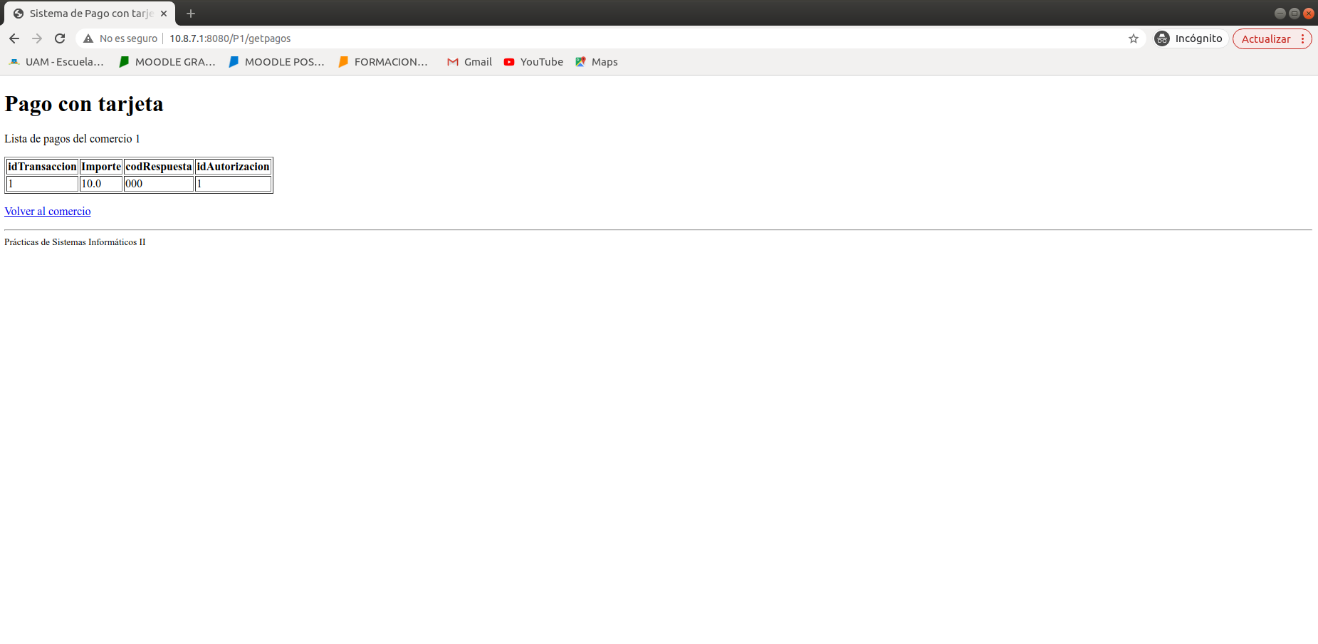
Para comprobar que el pago se ha realizado correctamente, buscamos en la base de datos la tabla de los pagos y comprobamos que se ha hecho.

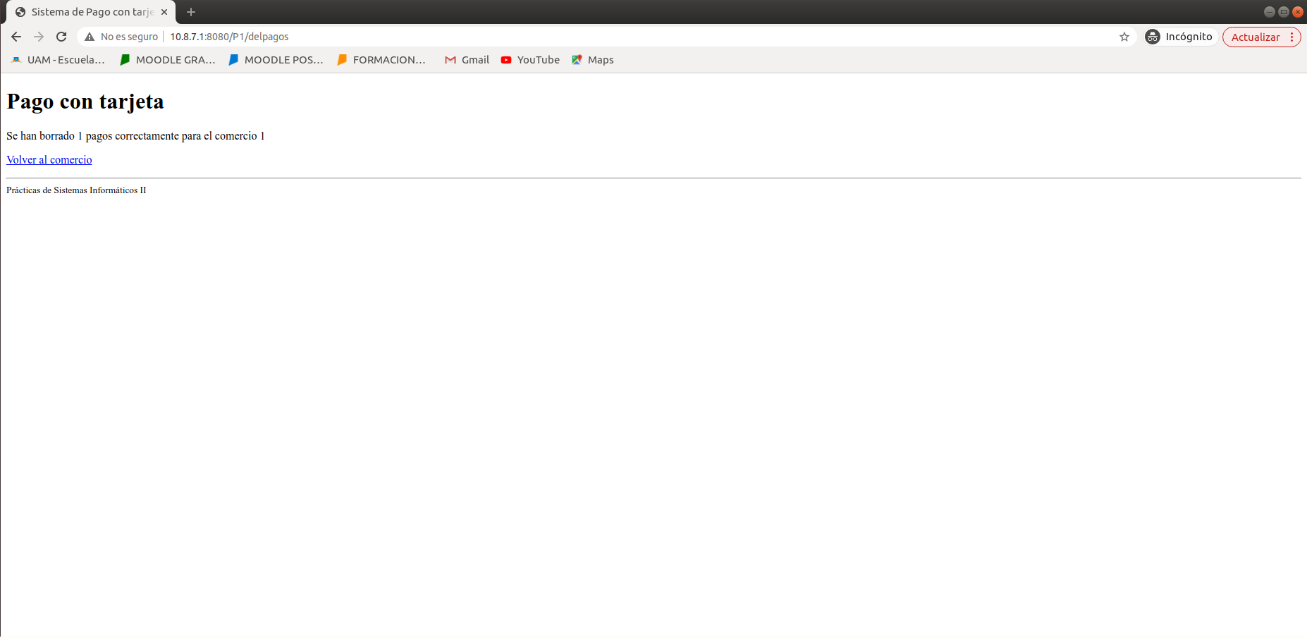


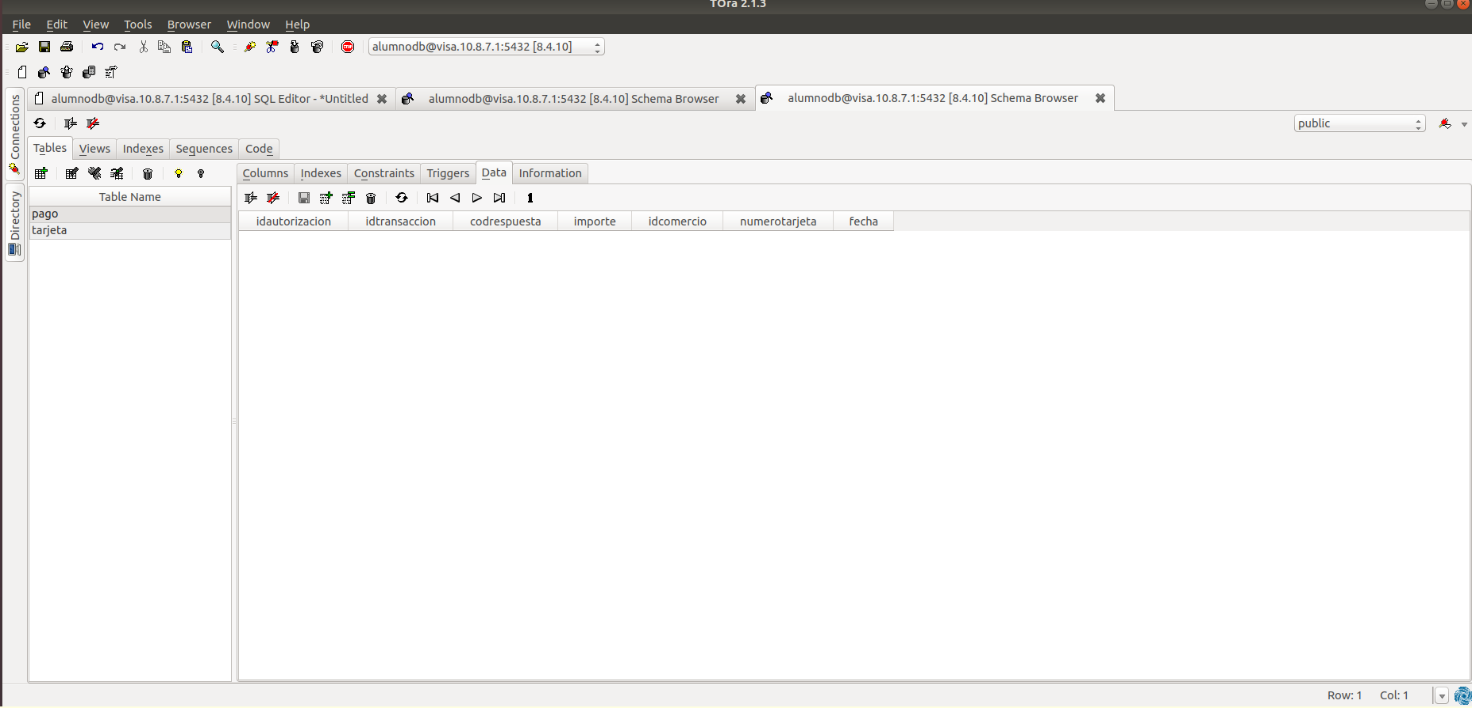
* **Acceda a la página de pruebas extendida, http://10.X.Y.Z:8080/P1/testbd.jsp. Compruebe que la funcionalidad de listado de y borrado de pagos funciona correctamente. Elimine el pago anterior.**

Primero, consultamos los pagos realizados al comercio 1 (que es la id que hemos utilizado al crear el pago en el apartado anterior). Nos redirige a una tabla en la que podemos ver nuestro pago, por lo que la funcionalidad de listado de pagos funciona correctamente.

A continuación, probamos la funcionalidad de borrado de pagos introduciendo de nuevo la misma id de comercio y vemos que nos redirige a una página en la que dice que se ha borrado un pago del comercio con la id que queríamos, por lo que parece que el borrado también funciona. Para comprobarlo del todo, abrimos la tabla de pagos de la base de datos desde TORA y comprobamos que está vacía. Por tanto, esta funcionalidad también funciona.

  Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente 



## Ejercicio número 2:

**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 posgresql, 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 poder realizar la conexión directa con la base de datos, hemos modificado, en el archivo DBTester.java, la variable JDBC\_DRIVER a “org.postgresql.Driver”, la variable JBC\_CONNSTRING a “jdbc:postgresql://10.8.7.1:5432/visa” para que incluya nuestro host (nuestra IP 10.8.7.1) y nuestro puerto (5432); y las variables JDBC\_USER a “alumnodb” y JDBC\_PASSWORD a “\*\*\*\*”, que son el usuario y la contraseña de nuestra base de datos.

Tras haber modificado el código, procedemos a la realización del pago de forma directa. Para ello, completamos los campos con la misma información que en el ejercicio 1 y seleccionamos los siguientes filtros:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Para comprobar que el pago se ha realizado correctamente, buscamos en la base de datos la tabla de los pagos y comprobamos que se ha hecho.

Una captura de pantalla de una red social

Descripción generada automáticamente

Primero, consultamos los pagos realizados al comercio 1 (que es la id que hemos utilizado al crear el pago en el apartado anterior). Nos redirige a una tabla en la que podemos ver nuestro pago, por lo que la funcionalidad de listado de pagos funciona correctamente.

A continuación, probamos la funcionalidad de borrado de pagos introduciendo de nuevo la misma id de comercio y vemos que nos redirige a una página en la que dice que se ha borrado un pago del comercio con la id que queríamos, por lo que parece que el borrado también funciona. Para comprobarlo del todo, abrimos la tabla de pagos de la base de datos desde TORA y comprobamos que está vacía. Por tanto, esta funcionalidad también funciona.

 Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamenteInterfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

## Ejercicio número 3:

**Examinar el archivo postgresql.properties para determinar el nombre del recurso JDBC correspondiente al DataSource y el nombre del pool.**

El recurso JDBC correspondiente al DataSource se identifica con “db.jdbc.resource.name=jdbc/VisaDB”, y el pool con “db.pool.name=VisaPool”. Ambos se encuentran en el archivo postgresql.properties.

**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.**

Accedemos a la consola de administración, comprobamos los recursos y el pool y realizamos el ping.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

**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.**

Añadimos a la memoria los valores que nos piden:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

**Comente razonadamente qué impacto considera que pueden tener estos parámetros en el rendimiento de la aplicación.**

Repercute en el sentido de que hay un número mínimo inicial de conexiones, que es 8, que va aumentando a medida que los clientes solicitan más. Pero tiene un máximo de conexiones disponibles, que son 32, por encima de estas, la base de datos no acepta más conexiones abiertas al mismo tiempo.

En ese momento, la base de datos irá eliminando las conexiones de dos en dos cuando excedan el tiempo de inactividad máximo (que es 300 segundos). Además, el tiempo máximo de espera para la conexión después de una solicitud son 60000 ms.

## Ejercicio número 4:

**Ejercicio 4. Localice los siguientes fragmentos de código SQL dentro del proyecto proporcionado (P1-base) correspondientes a los siguientes procedimientos:**

Todos los fragmentos de código que nos piden se encuentran en el archivo VISADAO.java,

* **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** (Texto de la respuesta)

Texto

Descripción generada automáticamente

## 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.**

**Incluya en la memoria capturas de pantalla del log del servidor, accediendo a él tanto desde el terminal como del portal web.**

Tras realizar el pago con la opción de debug, comprobamos el log del servidor desde el portal web.

Interfaz de usuario gráfica

Descripción generada automáticamente

Del mismo modo se puede obtener el log del servidor mediante la terminal accediendo al siguiente directorio:

/opt/glassfish-4.1.2/glassfish/domains/domain1/logs/server.log

Texto

Descripción generada automáticamente

## 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.**

Hemos organizado las carpetas de P1-ws como requiere la práctica, hemos cambiado el nombre del archivo VisaDAO.java a VisaDAOWS.java y cambiado las referencias al mismo en el resto de documentos. En el archivo VisaDAOWS.java añadimos los siguientes imports:

Texto

Descripción generada automáticamente

También, añadimos en la clase la siguiente etiqueta:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Para los siguientes métodos, hemos añadido las etiquetas que nos requerían en el pdf.

* **compruebaTarjeta()**

Texto

Descripción generada automáticamente

* **realizaPago()**

Texto

Descripción generada automáticamente

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

Texto

Descripción generada automáticamente

* **isPrepared() / setPrepared()**

Texto

Descripción generada automáticamente

**Deberemos publicar así mismo:**

* **isDirectConnection() / setDirectConnection()**

**que son métodos heredados de la clase DBTester.**

**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.**

Texto

Descripción generada automáticamente

Además, hemos modificado el resto de los métodos con las etiquetas. Si no queríamos que se publicaran directamente, en lugar de un operationName, hemos añadido exclude=true.

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

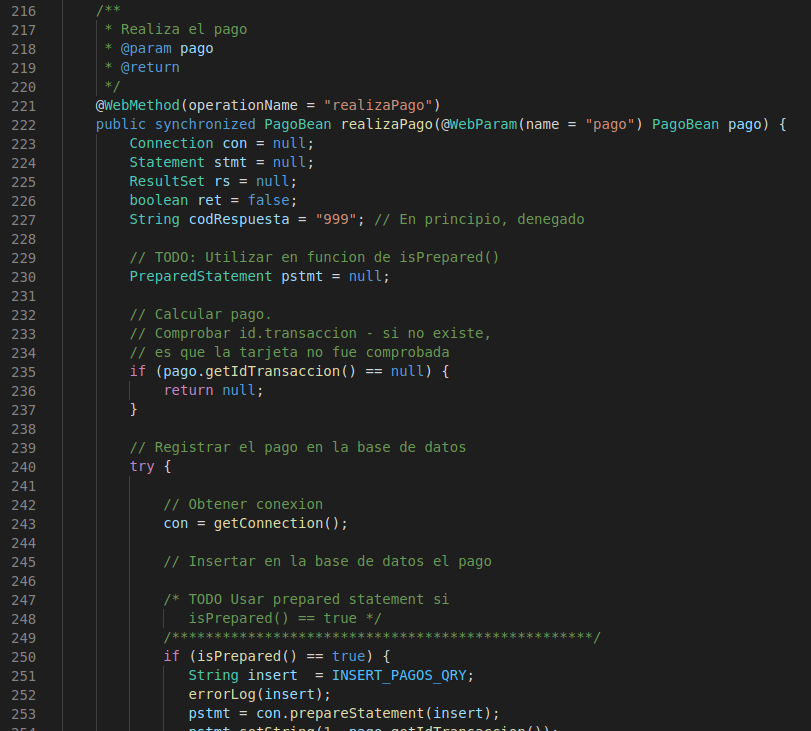
Texto

Descripción generada automáticamente

**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.**

Tras los cambios realizados en el método nuestro código es:



Texto

Descripción generada automáticamente

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

Al ser una estructura web service, trabajaremos en máquinas diferentes para el cliente y para servidor. Por tanto, como los argumentos van a pasarse por valor, el servidor tiene que devolver el objeto tras su modificación.

## Ejercicio número 7:

**Texto de la cuestión**

**Despliegue el servicio con la regla correspondiente en el build.xml.**

Para desplegar el servicio, utilizamos el comando “ant compilar-servicio empaquetar-servicio desplegar-servicio”. Después, accedemos con GlassFish a nuestras aplicaciones y vemos que aparece desplegada correctamente.

Interfaz de usuario gráfica, Texto, Correo electrónico

Descripción generada automáticamente

**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).**

Para acceder remotamente, encontramos el enlace en la información del endpoint de nuestro web service.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Accedemos al enlace a través del navegador y comprobamos que la URL contiene la IP de nuestra máquina virtual, que es 10.8.7.1.

Interfaz de usuario gráfica, Texto, Correo electrónico

Descripción generada automáticamente

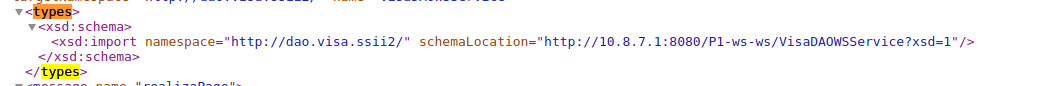
**Comente en la memoria aspectos relevantes del código XML del fichero WSDL y su relación con los métodos Java del objeto del servicio, argumentos recibidos y objetos devueltos.**

Los aspectos relevantes del código son aquellos contenidos dentro de las etiquetas del documento XML. Por tanto, dentro de la etiqueta types tenemos el enlace a la url que contiene los tipos, los métodos y los atributos de estos métodos; dentro de la etiqueta message, podemos ver los métodos que se ejecutan; en la etiqueta portType podemos ver los inputs y outputs de los métodos que se ejecutan.

**Conteste a las siguientes preguntas:**

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

Los tipos de datos intercambiados con el webservice se definen en el fichero que se referencia en la etiqueta types.



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

Los datos predefinidos han sido identificados porque llevan antes del tipo "xs:". Por tanto, hemos identificado boolean, string y double.

Texto

Descripción generada automáticamente

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

Los datos predefinidos han sido identificados porque llevan antes del tipo "tns:". Por tanto, hemos identificado tarjetaBean, pagoBean, compruebaTarjeta, compruebaTarjetaResponse, isDebug, isDebugResponse, isDirectConnection, isDirectConnectionResponse, isPrepared, isPreparedResponse, realizaPago, realizaPagoResponse, setDebug, setDebugResponse, setDirectConnection, setDirectConnectionResponse, setPrepared, setPreparedResponse.

Texto

Descripción generada automáticamente

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

La etiqueta <messages>.

Tabla

Descripción generada automáticamente

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

En la etiqueta <portType>.

Texto

Descripción generada automáticamente

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

En la etiqueta <binding>.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

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

En la etiqueta <service>.

Texto

Descripción generada automáticamente con confianza media

## 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:**

* **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.**

Todas las modificaciones se realizan en el archivo pago.java. El nuevo código nos quedaría:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamenteTexto

Descripción generada automáticamenteCaptura de pantalla de un celular

Descripción generada automáticamente

## 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.**

Modificamos el archivo procesapago.java.

Pantalla de computadora con fondo negro

Descripción generada automáticamente con confianza baja

Texto

Descripción generada automáticamente

## 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:**

* **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.**

Las siguientes modificaciones se han hecho en el archivo getPagos.java.

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

## 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. Téngase en cuenta que el servicio debe estar previamente desplegado.**

Para realizar la importación, utilizamos el siguiente comando:

Texto

Descripción generada automáticamente

Las clases generadas son:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Se han generado las clases que habíamos incluido en el documento WSDL del servicio. Se generan porque son los objetos intermedios antes de que el servidor se los envíe al cliente.

## 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}**

Completamos el target en el archivo build.xml:

Texto

Descripción generada automáticamente

## 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.**

**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.**

Para comprobar que el servidor se ejecuta correctamente procedemos a ejecutar pagos desde el cliente.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Comenzamos haciendo un pago desde el html obteniendo el siguiente resultado:

Continuamos comprobando el correcto funcionamiento del pago ejecutándolo desde testbd.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Tras comprobar el funcionamiento de los pagos procedemos a comprobar que se han guardado correctamente.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Por último, eliminamos los pagos realizados.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

## Cuestión número 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.**

1. Por parte del comercio se genera una petición de pago.
2. El servlet comienzaPago recibe esta petición y comprueba los datos para que sean correctos.
3. Se muestra un formulario al cliente para que complete los datos de su tarjeta.
4. Los datos se envían al servlet ProcesaPago, que comprueba que sean correctos.
5. En el momento en el que se comprueba la caducidad de la tarjeta y se ve que no es válida porque ya ha pasado, se vuelve a mostrar el formulario y aparece un error llamando a error/muestraerror.jsp.

## Cuestión número 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?**

Antes de obtener la información sobre el pago con tarjeta, primero se obtiene la información sobre el comercio y el importe en el que se va a realizar dicho pago con el servlet ComienzaPago Los datos que se obtienen en este formulario son la id de la transacción, la id del comercio, el importe y la ruta.

Después, una vez que esos datos son correctos, el servlet encargado de obtener la información sobre el pago con tarjeta es ProcesaPago porque es la clase en la que se obtienen a través de un formulario el nombre del titular de la tarjeta, número de tarjeta, fecha de caducidad, fecha de emisión y CVV2. Además, esta clase también procesa esta información, porque una vez que la obtiene, llama al método compruebaTarjeta().

## Cuestión número 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?**

La instancia de la clase pago cuando se accede por pago.html se crea utilizando el servlet ComienzaPago. En cambio, cuando se accede por testbd.jsp la instancia de la clase pago se crea mediante el servlet ProcesaPago directamente. La información se comparte a través de los servlets con HttpServletRequest y HttpServletResponse. La información se almacena en la Bean y esta se crea en el propio servlet.

## Cuestión número 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 el caso de pago.html, se llama al servlet ComienzaPago el cual llama a validaParametros con la información de la tarjeta que se le ha introducido al formulario, después se reenvía a formdatosvisa.jsp donde con la Bean se rellenan los datos restantes, se validan y se llama finalmente al servlet ProcesaPago. Por otro lado, testbd.jsp pide directamente todos los datos y procede a llamar a ProcesaPago directamente. Esto funciona ya que en el caso de que no haya una instancia de pago creada, en el mismo ProcesaPago, se crea una nueva.