

# MEMORIA PRÁCTICA 1A

## Arquitectura Java EE

Alejandro Santorum Varela - alejandro.santorum@estudiante.uam.es

Rafael Sánchez Sánchez - rafael.sanchezs@estudiante.uam.es

Prácticas Sistemas Informáticos II

Práctica 1A Pareja 7 Grupo 2401

24 de febrero de 2020

## Contents

<b>I</b>	<b>Introducción</b>	<b>2</b>
<b>II</b>	<b>Configuración</b>	<b>2</b>
<b>1</b>	<b>Ejercicio 1</b>	<b>2</b>
<b>2</b>	<b>Ejercicio 2</b>	<b>11</b>
<b>3</b>	<b>Ejercicio 3</b>	<b>15</b>
<b>4</b>	<b>Ejercicio 4</b>	<b>17</b>
<b>5</b>	<b>Ejercicio 5</b>	<b>18</b>
<b>6</b>	<b>Ejercicio 6</b>	<b>25</b>
<b>7</b>	<b>Ejercicio 7</b>	<b>27</b>
<b>8</b>	<b>Ejercicio 8</b>	<b>32</b>
<b>9</b>	<b>Ejercicio 9</b>	<b>32</b>
<b>10</b>	<b>Ejercicio 10</b>	<b>33</b>
<b>11</b>	<b>Ejercicio 11</b>	<b>35</b>
<b>12</b>	<b>Ejercicio 12</b>	<b>36</b>
<b>13</b>	<b>Ejercicio 13</b>	<b>37</b>
<b>14</b>	<b>Cuestiones</b>	<b>41</b>
14.1	Cuestión 1 . . . . .	41
14.2	Cuestión 2 . . . . .	41
14.3	Cuestión 3 . . . . .	41
14.4	Cuestión 4 . . . . .	41
<b>III</b>	<b>Conclusiones</b>	<b>42</b>
<b>IV</b>	<b>Bibliografía</b>	<b>42</b>

# I Introducción

Nos encontramos en la primera práctica del curso de Sistemas Informáticos II. El objetivo fundamental de esta práctica es adentrarse en la arquitectura de JAVA EE desde el punto de vista del arquitecto de software.

A continuación se muestran los resultados obtenidos y las respuestas a las preguntas solicitadas.

## II Configuración

Antes de intentar realizar cualquier ejercicio deberemos configurar el ordenador del laboratorio para poder ejecutar los ficheros necesarios:

1. Seleccionar la imagen importada (si2srv).
2. Generar MAC *address* del Network Adapter (NAT) y Network Adapter 2 (Bridged) de la máquina virtual (en *settings*).
3. Nos *logeamos* con usuario **si2** y contraseña **2020sid0s**.
4. La primera vez que entramos en la máquina virtual deberemos configurar una dirección IP única y sin conflictos en el rango 10.X.Y.Z. En nuestro caso **10.1.7.Z** donde  $Z \in \{1, 2, 3, 4\}$ .
5. Ahora, para acceder de forma remota, será necesario asignar a la interfaz de red del Host la dirección IP en el rango 10.X.Y.Z. En nuestro caso **10.1.7.Z** donde  $Z \in \{1, 2, 3, 4\}$ .
6. Definiremos la variable de entorno de Glassfish:  
`export J2EE_HOME=/opt/glassfish4/glassfish` (en una terminal del Host).
7. A continuación podremos acceder de forma remota, con un terminal del host, usando el comando `ssh si2@10.1.7.Z`.
8. Finalmente, podremos iniciar el servidor de Glassfish con el comando:  
`asadmin start-domain domain1`.

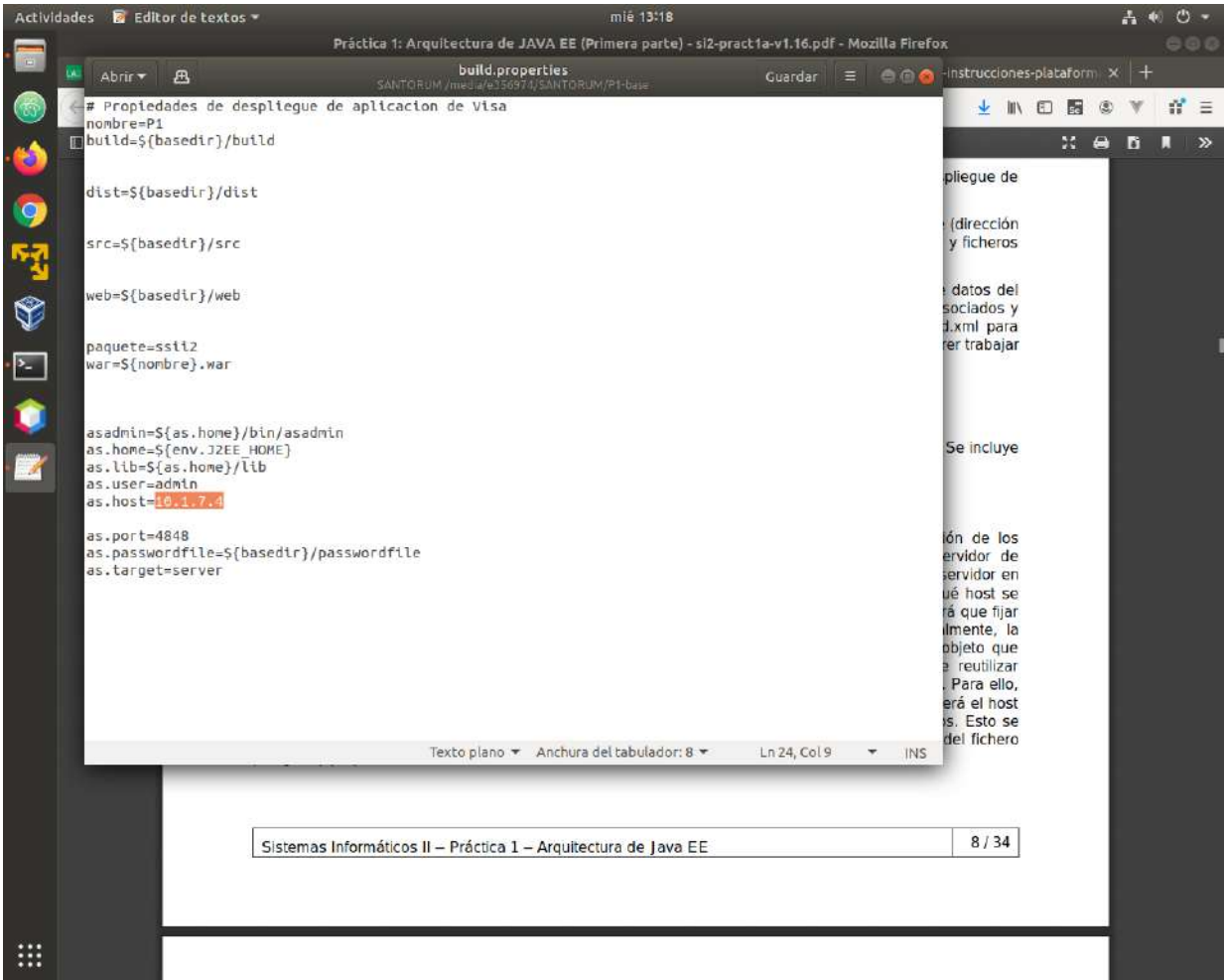
## 1 Ejercicio 1

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

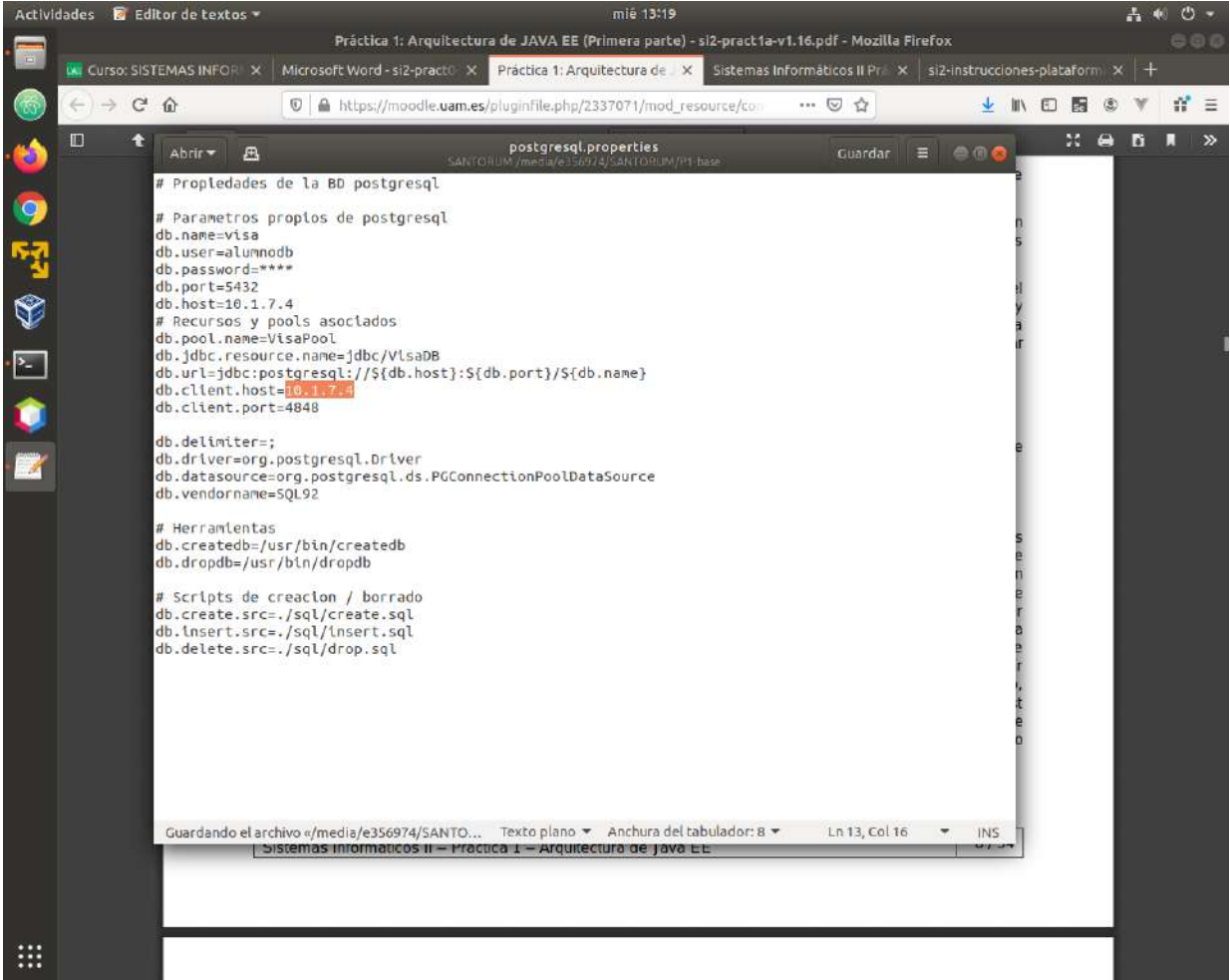
1. Modifique los ficheros que considere necesarios en el proyecto para que se despliegue tanto la aplicación web como la base de datos contra la dirección asignada a la pareja de prácticas.
2. Realice un pago contra la aplicación web empleando el navegador en la ruta:  
`http://10.X.Y.Z:8080/P1`
3. Conéctese a la base de datos (usando el cliente Tora por ejemplo) y obtenga evidencias de que el pago se ha realizado.
4. 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.

Respuesta a la cuestión:

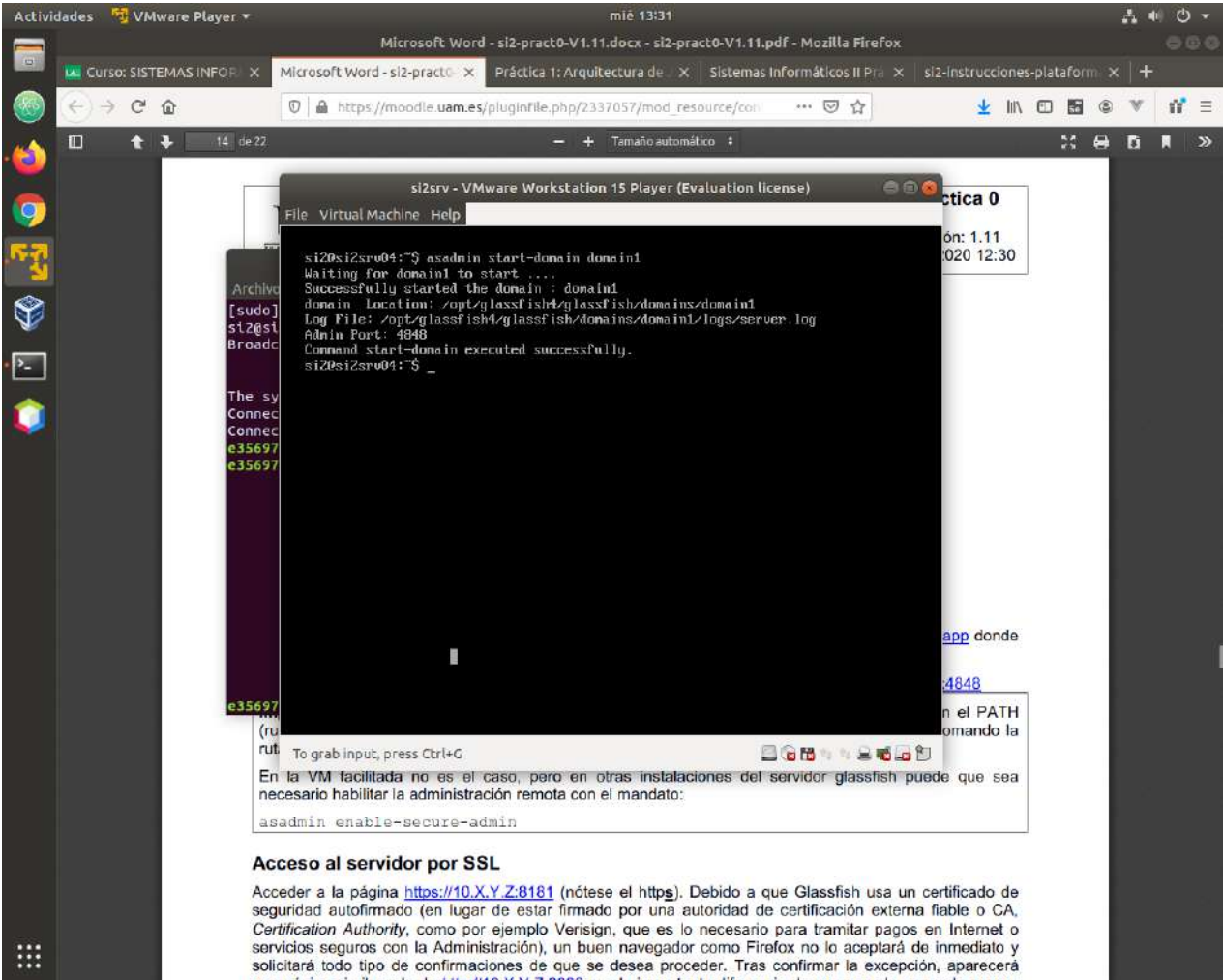
Empezamos modificando los ficheros necesarios para arrancar la aplicación web. En primer lugar modificaremos el fichero `build.properties` en la línea donde se especifica `as.host` y lo asignaremos a la IP `10.1.7.4`.



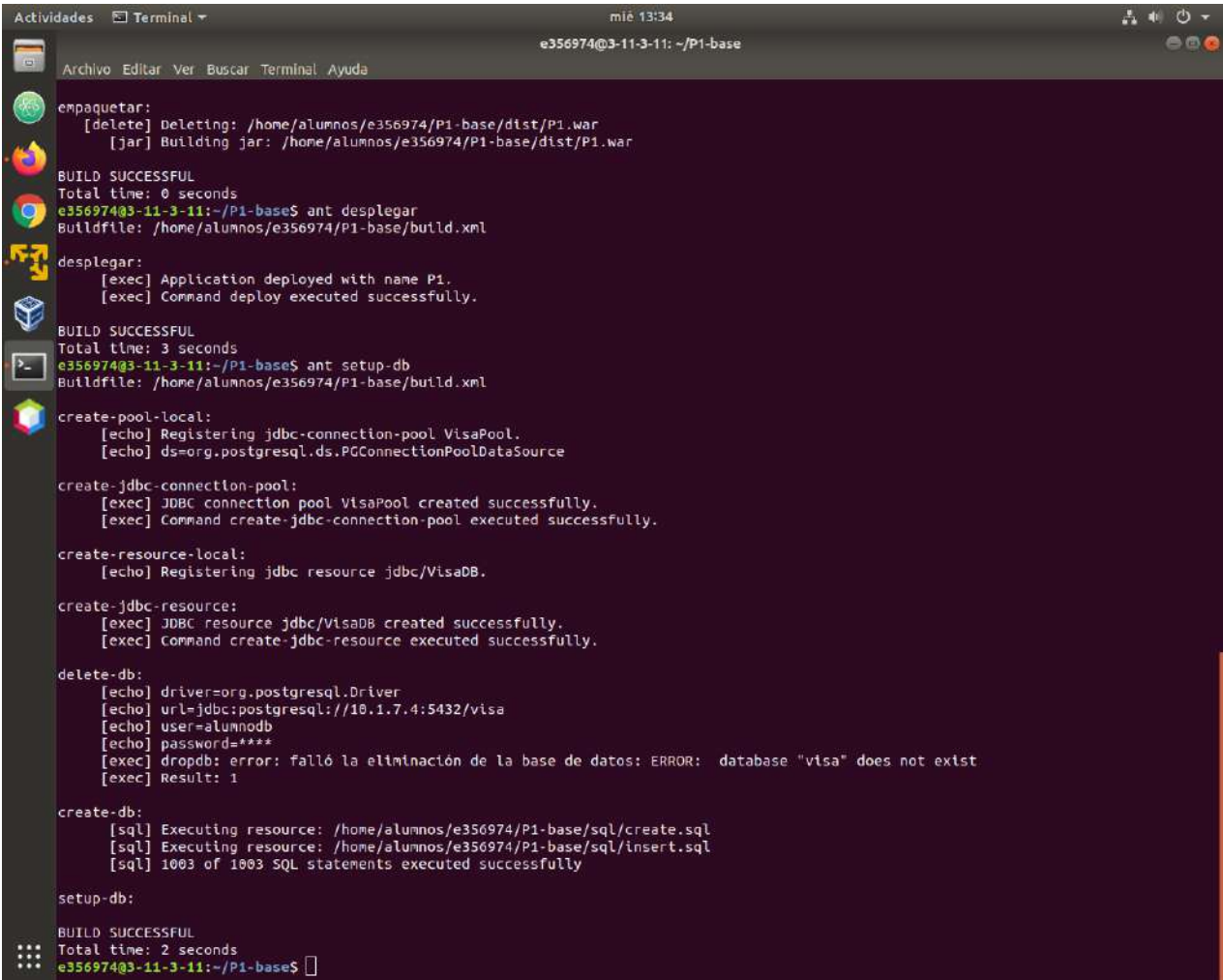
Haremos lo mismo con el fichero `postgresql.properties`, en la variable `db.host` y en la variable `db.client.host`



Ahora deberemos arrancar el servidor con los nuevos cambios realizados

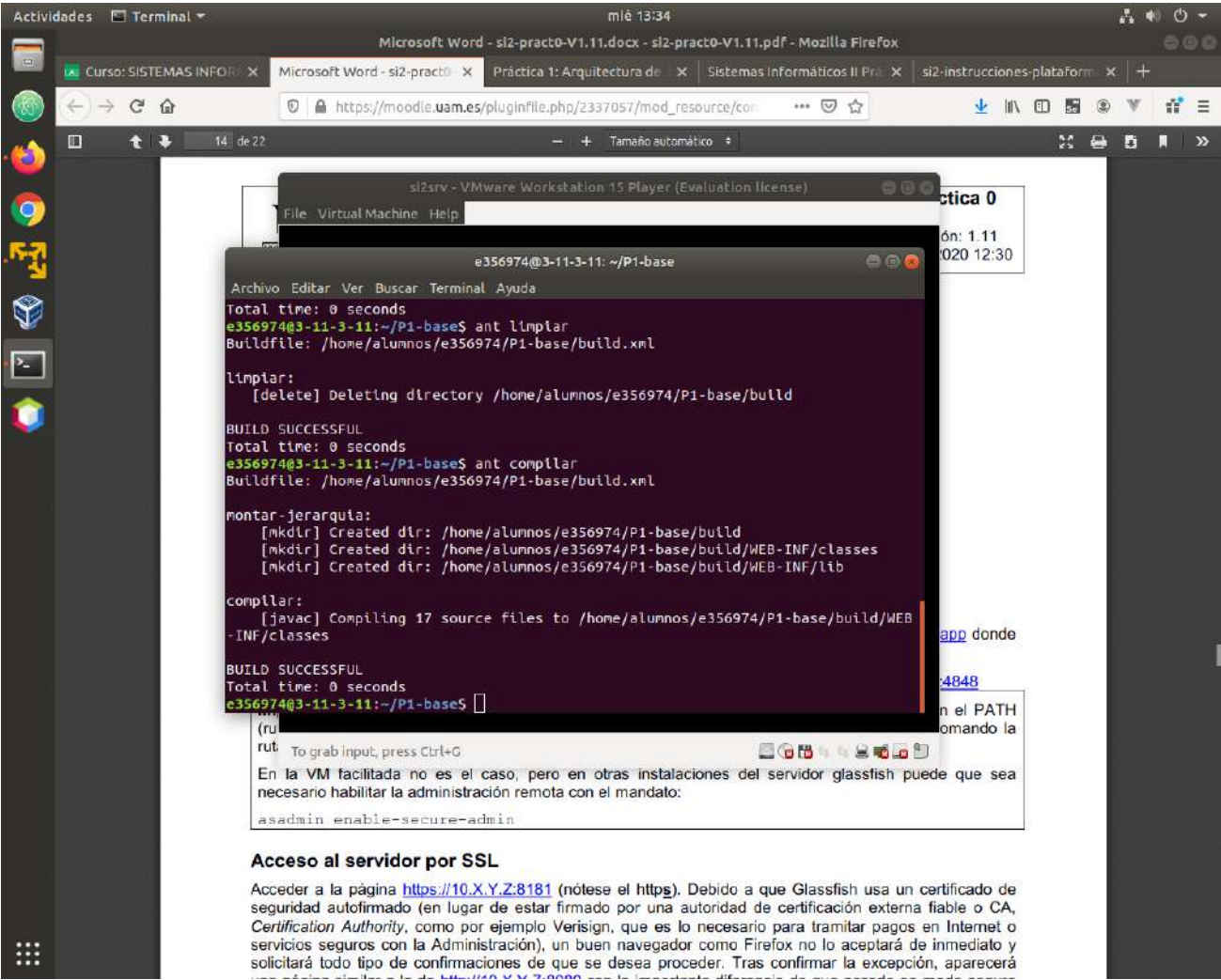


A continuación deberemos compilar todos los ficheros necesarios para el despliegue de la aplicación web. Empezamos iniciando la base de datos con el comando **ant setup-db** (dentro del directorio P1-base, donde se encuentran los ficheros fuente de la app).

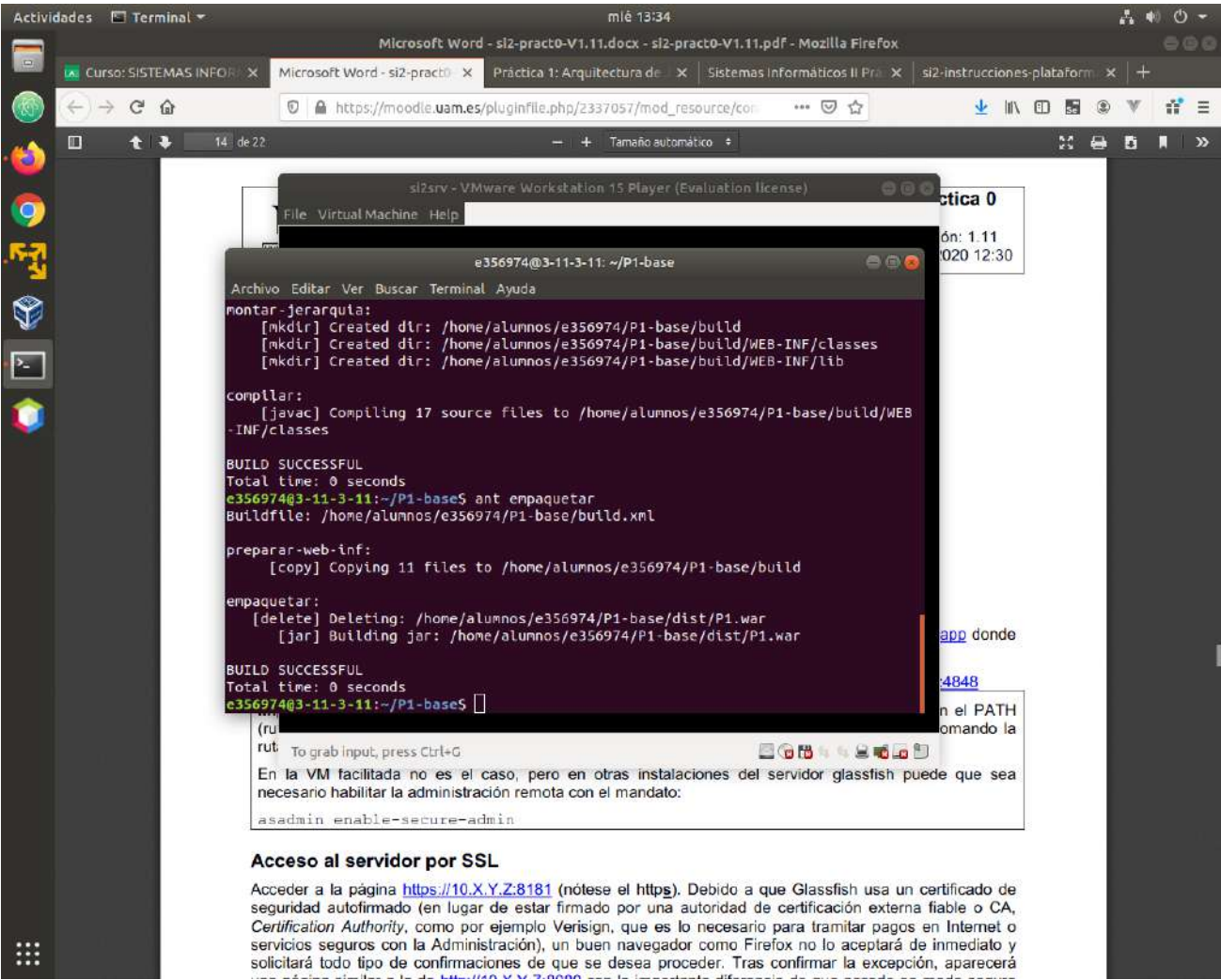




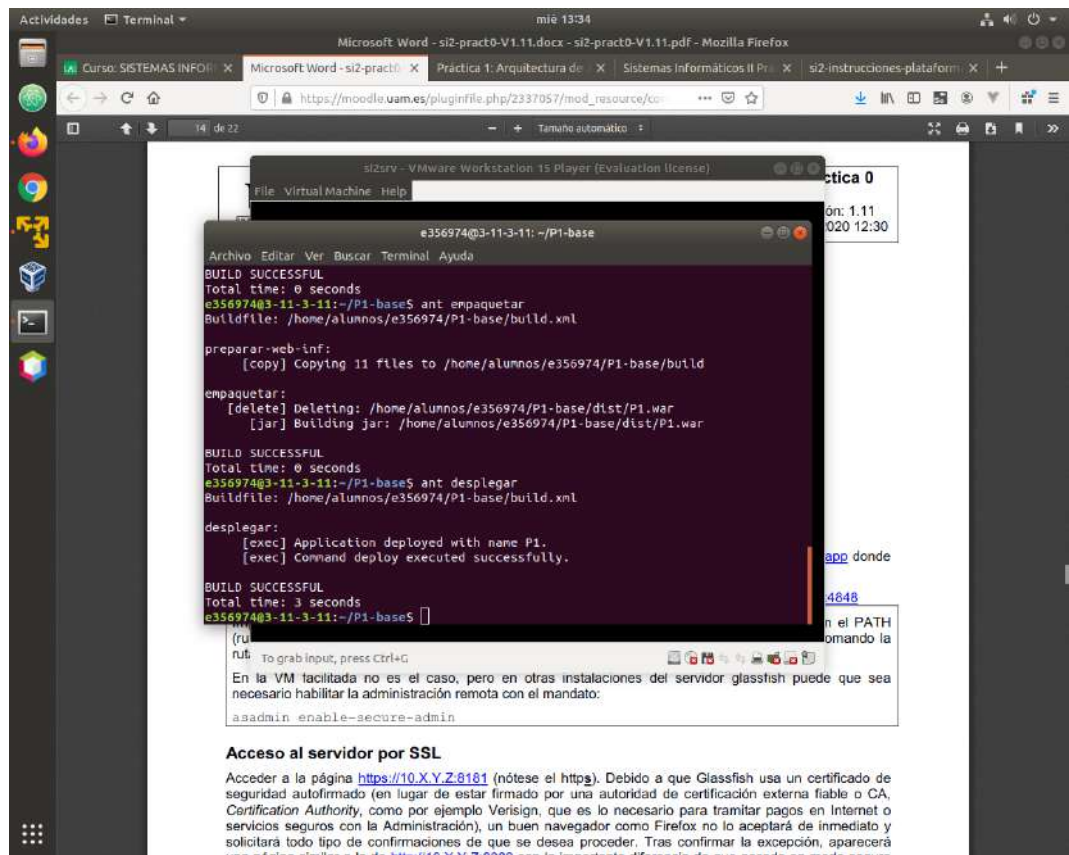
Después, compilamos los ficheros necesarios con **ant** **compilar**



Y preparamos del paquete con la aplicación web (fichero .war) con **ant** **empaquetar**.

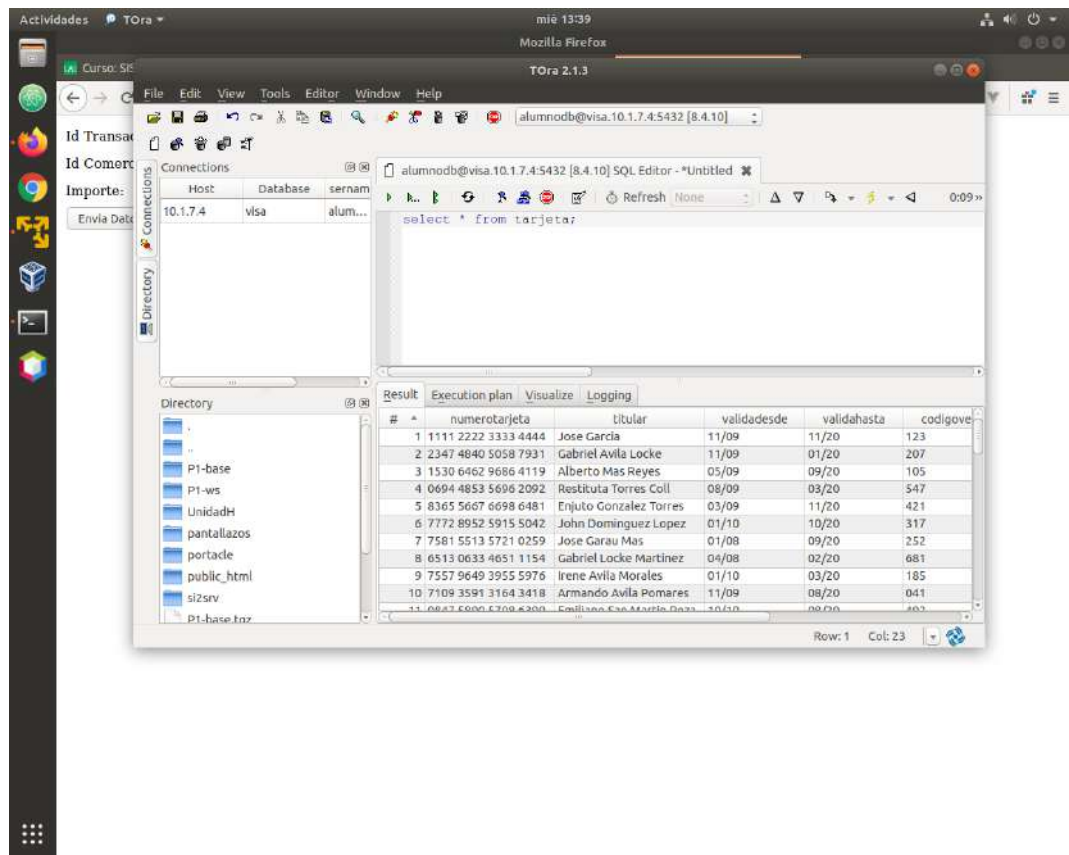


Finalmente, podemos desplegar la aplicación con **ant** desplegar



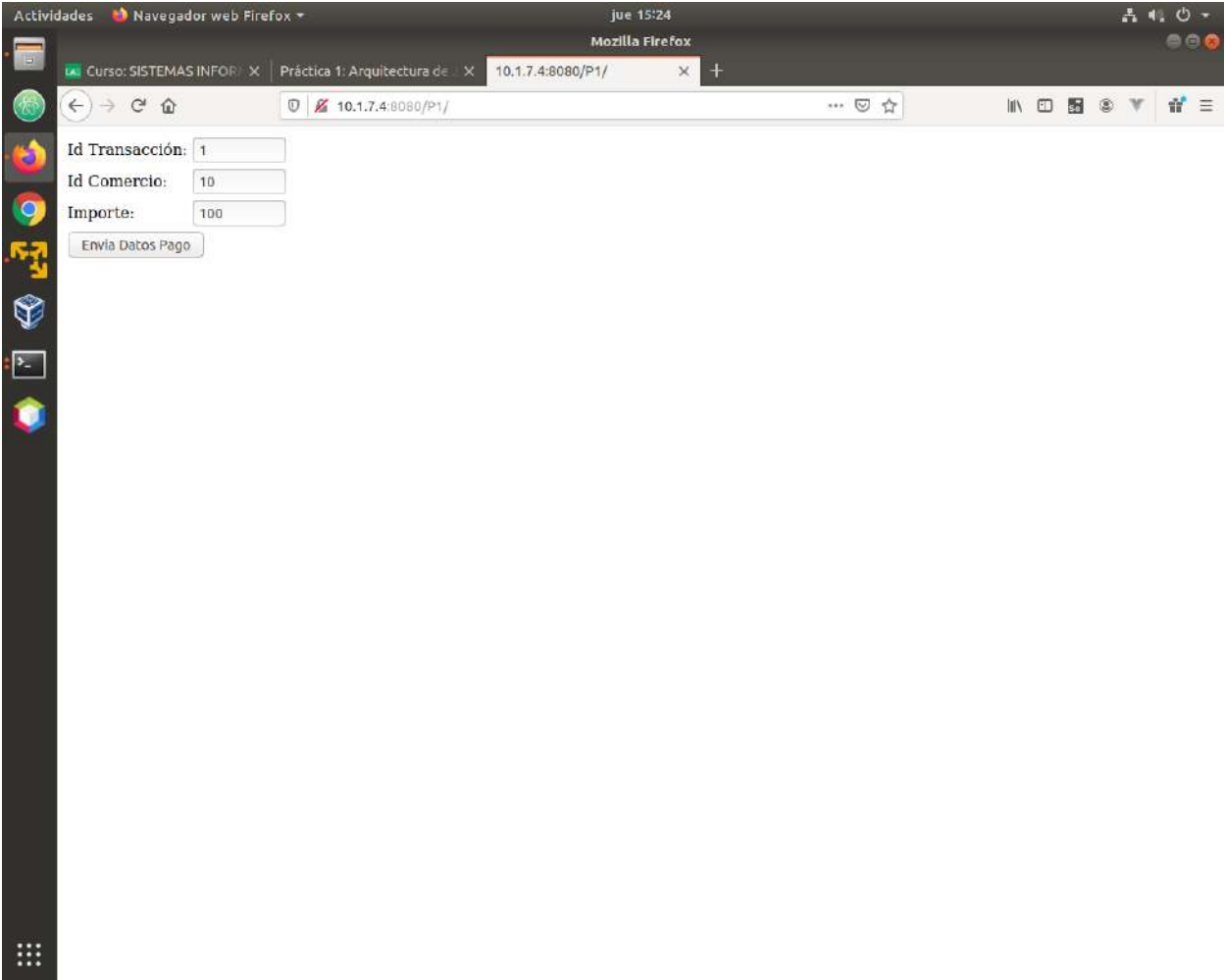
Comentar que toda esta secuencia de comandos se puede abreviar por el comando **ant todo**, que se encarga de ejecutar secuencialmente los comandos **ant** descritos anteriormente. Ahora, con la aplicación lanzada, es el momento de intentar realizar un pago en ella. Para ello deberemos conocer de mano una tarjeta de crédito válida registrada en la base de datos, así como el resto de datos del cliente. Esta información la podemos obtener de dos maneras básicas:

1. Con la aplicación Tora, conectándose a la base de datos de forma remota y comprobando la tabla **tarjeta**.



2. Con la terminal conectada a la máquina virtual (usando ssh) podremos ejecutar el comando **psql visa -U alumnodb** para conectarse a la base de datos y finalmente con la consulta SQL **select \* from tarjeta** podremos obtener la misma información que en el punto 1.

Una vez recolectados estos datos, podemos ya realizar un pago. Con un navegador web entraremos en la ruta **http://10.1.7.4:8080/P1** e introduciremos los datos de la transacción.

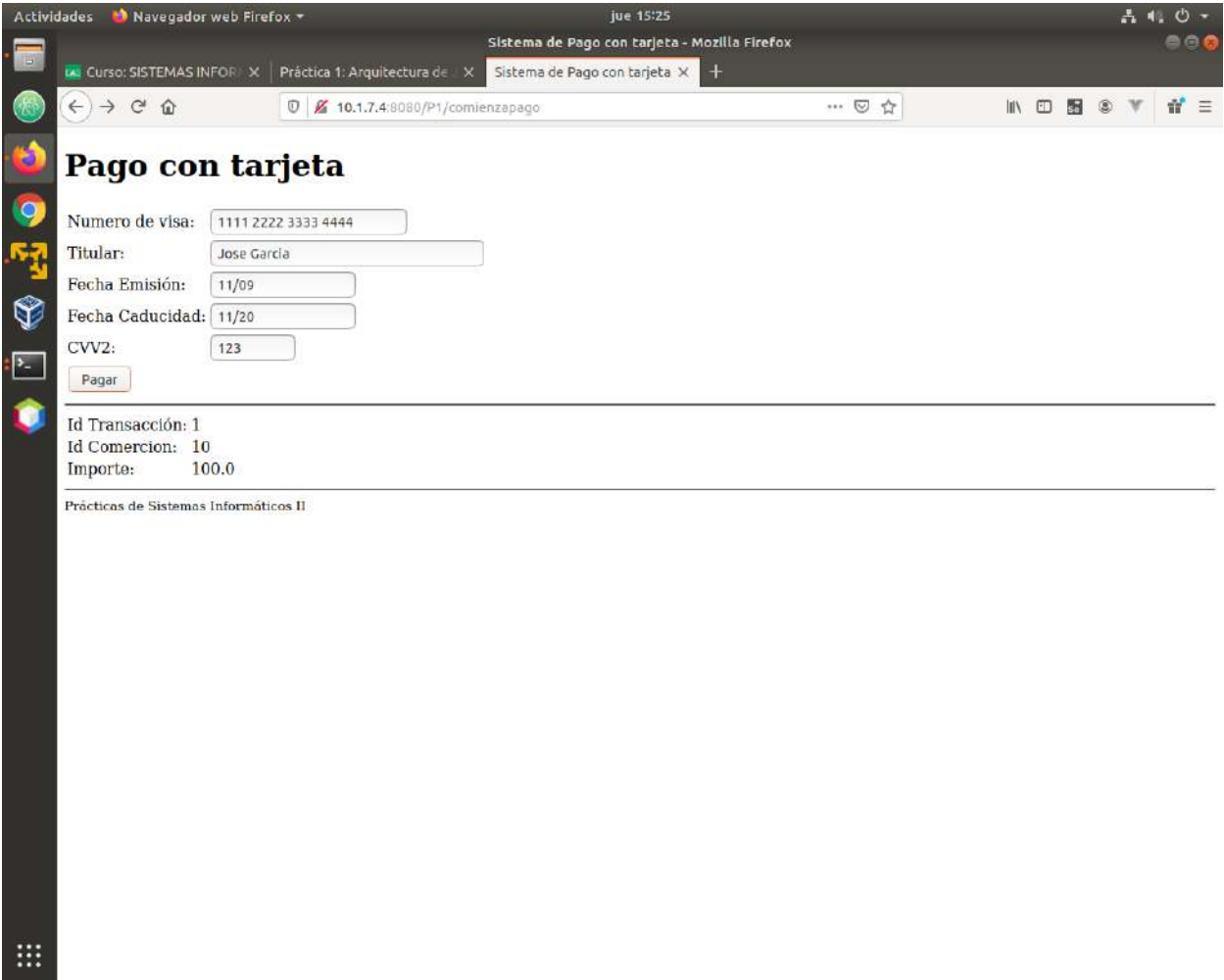


The screenshot shows a Mozilla Firefox browser window with the address bar displaying `10.1.7.4:8080/P1/`. The page contains a form with the following fields and values:

Field	Value
Id Transacción:	1
Id Comercio:	10
Importe:	100

Below the form is a button labeled "Envia Datos Pago".

Pulsando en 'Envia Datos Pago' llegaremos a otro formulario donde introduciremos los datos de un cliente cualquiera obtenidos anteriormente.



The screenshot shows the same Mozilla Firefox browser window, but the address bar now displays `10.1.7.4:8080/P1/comienzapago`. The page title is "Sistema de Pago con tarjeta - Mozilla Firefox". The form is titled "Pago con tarjeta" and contains the following fields and values:

Field	Value
Numero de visa:	1111 2222 3333 4444
Titular:	Jose Garcia
Fecha Emisión:	11/09
Fecha Caducidad:	11/20
CVV2:	123

Below the card details is a button labeled "Pagar".

Below the "Pagar" button, the following transaction details are displayed:

Field	Value
Id Transacción:	1
Id Comercion:	10
Importe:	100.0

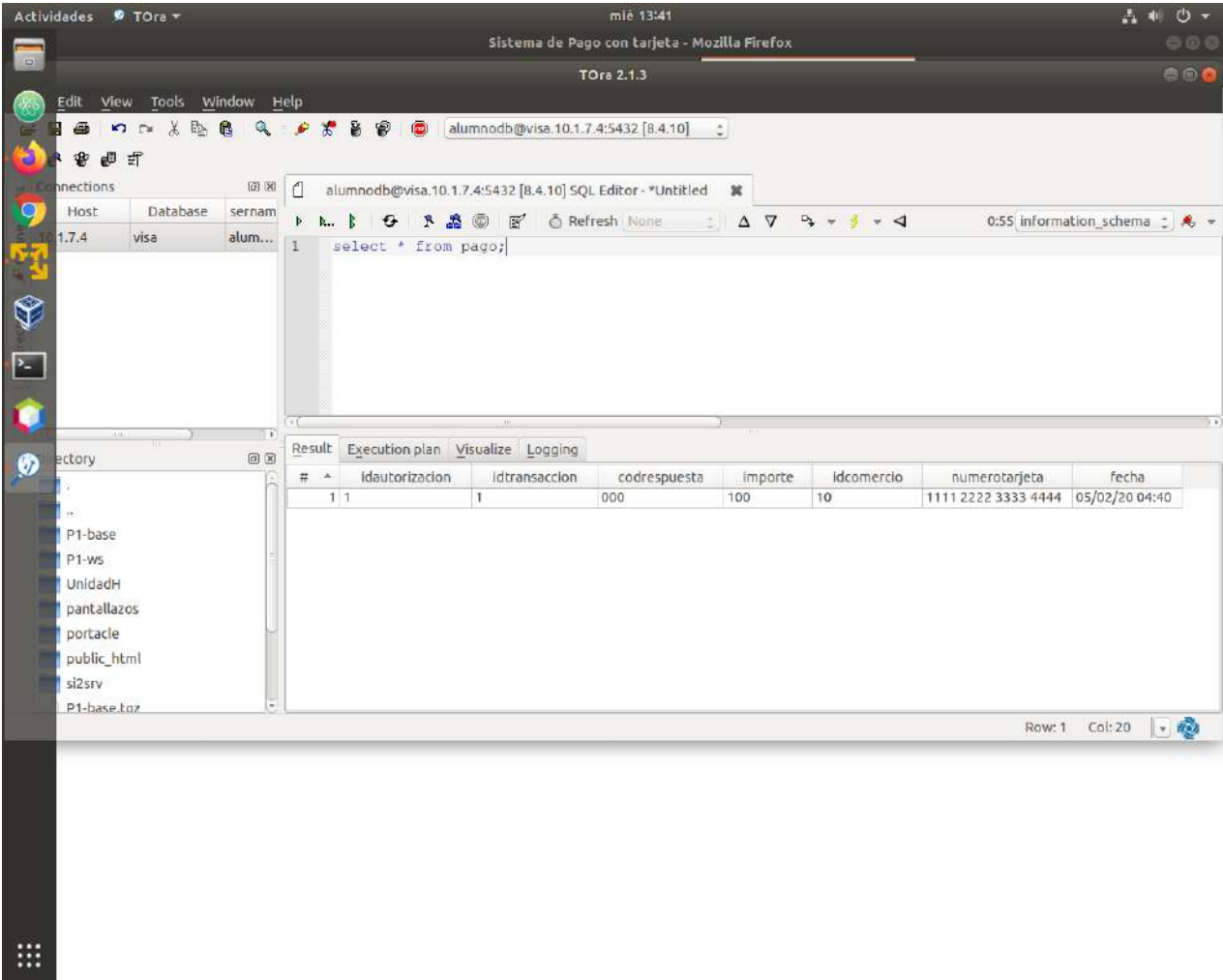
At the bottom of the page, the text "Prácticas de Sistemas Informáticos II" is visible.



Pulsando en 'Pagar' y si hemos realizado todo según se ha descrito obtendremos un pago exitoso.



Podemos comprobarlo en la base de datos (con Tora o con PostgreSQL):





Y también podremos comprobarlo en la página de pruebas extendida:  
`http://10.1.7.4:8080/P1/testbd.jsp`  
Escribiendo en el apartado 'Consulta de pagos' el ID del comercio especificado en la transacción.

Actividades

Navegador web Firefox

mié 13:42

Sistema de Pago con tarjeta - Mozilla Firefox

Curso: SISTEMAS INFOR X Microsoft Word - si2-pract X Práctica 1: Arquitectura de X Sistema de Pago con tarjeta X Sistema de Pago con tarjeta X +

10.1.7.4:8080/P1/testbd.jsp

# 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

Pagar

## Consulta de pagos

Id Comercio:

10

GetPagos

## Borrado de pagos

Id Comercio:

DelPagos

Prácticas de Sistemas Informáticos II

Actividades

Navegador web Firefox

mié 13:42

Sistema de Pago con tarjeta - Mozilla Firefox

Curso: SISTEMAS INFOR X Microsoft Word - si2-pract X Práctica 1: Arquitectura de X Sistema de Pago con tarjeta X Sistema de Pago con tarjeta X +

10.1.7.4:8080/P1/getpagos

# Pago con tarjeta

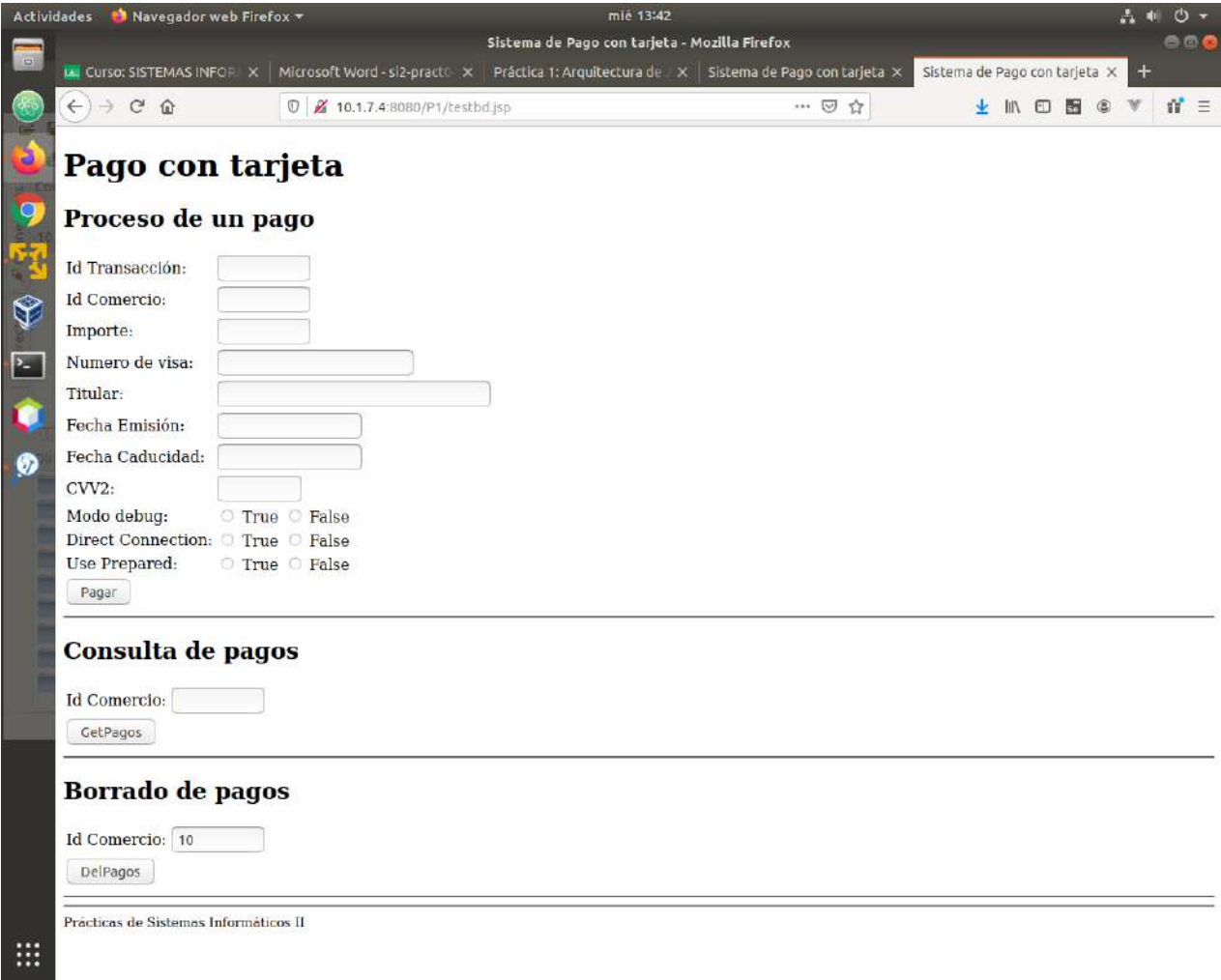
Lista de pagos del comercio 10

idTransaccion	Importe	codRespuesta	idAutorizacion
1	100.0	000	1

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

Finalmente podremos borrar el pago realizado desde la misma página de pruebas en el apartado 'Borrado de pagos', escribiendo el ID del comercio especificado en la transacción.



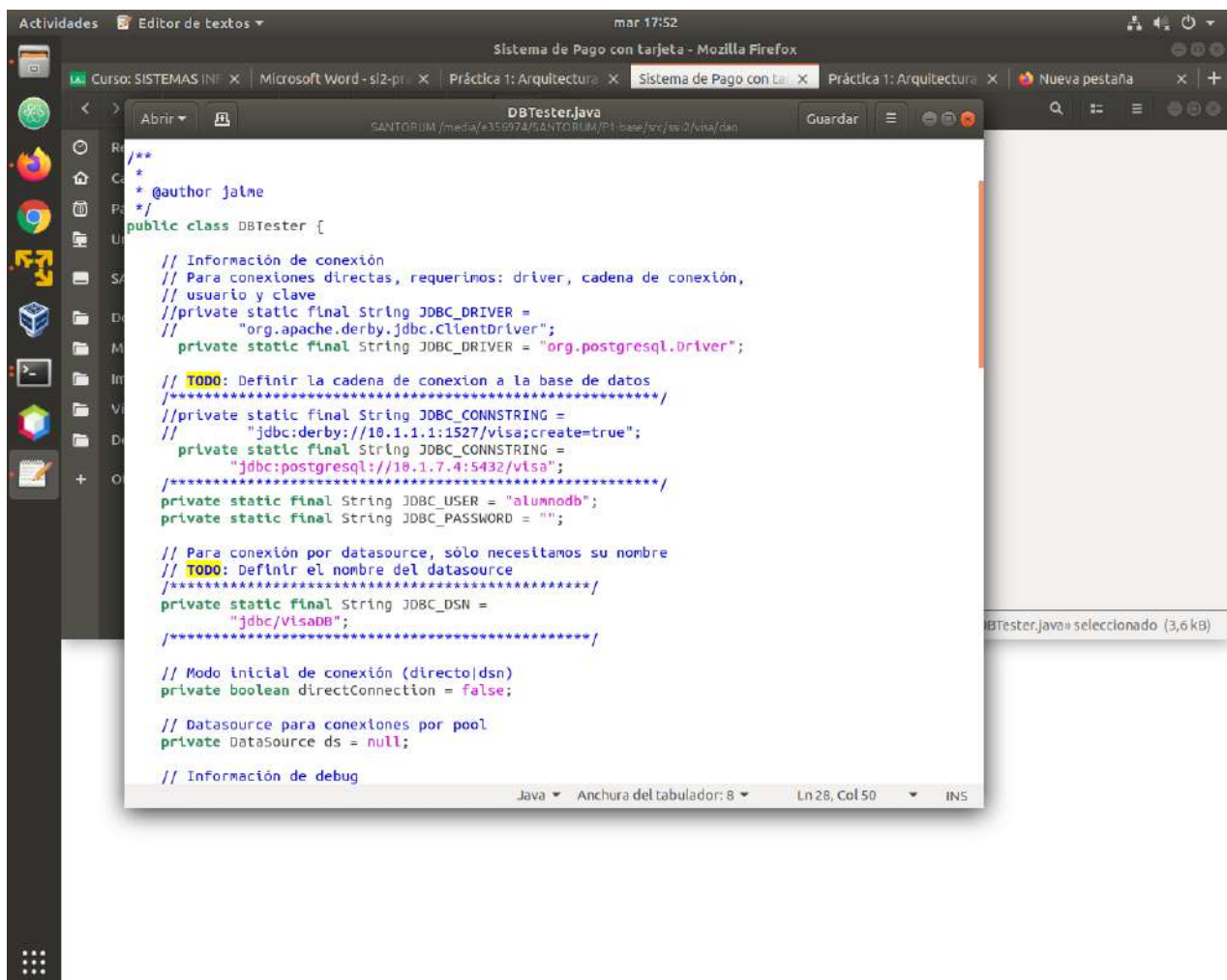
## 2 Ejercicio 2

**Enunciado:** 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.

### Respuesta a la cuestión:

Empezamos modificando el fichero **DBTester.java** localizado en **P1-base/src/ssii2/visa/dao**. Modificamos las siguientes variables, que estaban configuradas de forma incorrecta para una conexión directa:

- JDBC\_DRIVER: cambiamos "org.apache.derby.jdbc.ClientDriver" por "org.postgresql.Driver".
- JDBC\_CONNSTRING: cambiamos "jdbc:derby://10.1.1.1:1527/visa;create=true" por "jdbc:postgresql://10.1.7.4:5432/visa".
- JDBC\_USER debe ser igual a "alumnodb".
- JDBC\_PASSWORD es indiferente, por ejemplo "".



```
/**
 *
 * @author jalme
 */
public class DBTester {

    // Información de conexión
    // Para conexiones directas, requerimos: driver, cadena de conexión,
    // usuario y clave
    //private static final String JDBC_DRIVER =
    // "org.apache.derby.jdbc.ClientDriver";
    //private static final String JDBC_DRIVER = "org.postgresql.Driver";

    // TODO: Definir la cadena de conexión a la base de datos
    //private static final String JDBC_CONNSTRING =
    // "jdbc:derby://10.1.1.1:1527/visa;create=true";
    //private static final String JDBC_CONNSTRING =
    // "jdbc:postgresql://10.1.7.4:5432/visa";

    //private static final String JDBC_USER = "alumnodb";
    //private static final String JDBC_PASSWORD = "";

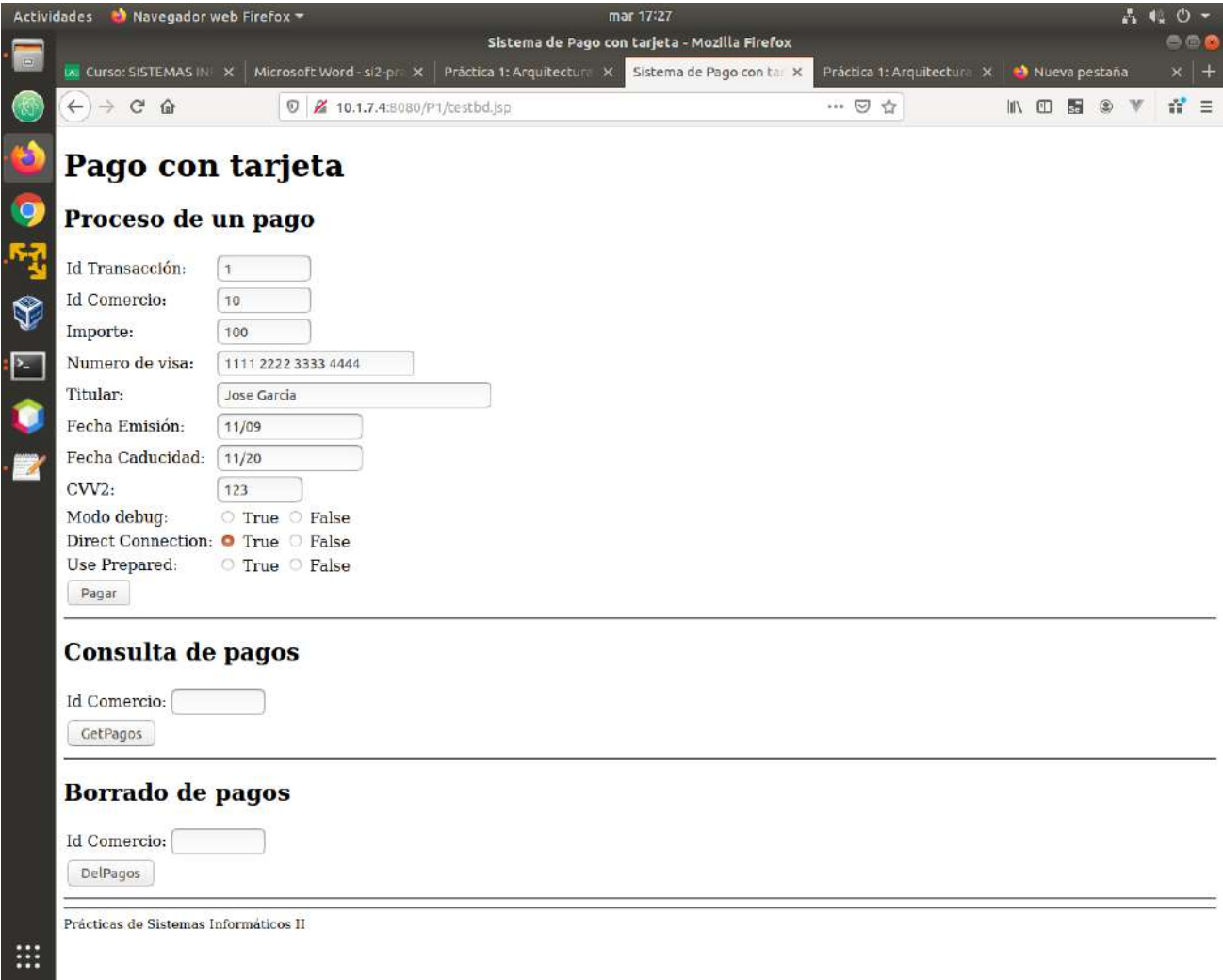
    // Para conexión por datasource, sólo necesitamos su nombre
    // TODO: Definir el nombre del datasource
    //private static final String JDBC_DSN =
    // "jdbc/visaDB";

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

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

    // Información de debug
```

Ahora ya podemos acceder a la página de pruebas extendida <http://10.1.7.4:8080/P1/testbd.jsp> para probar a realizar un pago con conexión directa. Para ello utilizaremos los datos usados en el ejercicio 1, y seleccionaremos la opción **Direct Connection**.



Si todo ha ido bien, deberemos obtener el siguiente mensaje de confirmación. Pulsaremos en 'Volver al comercio' para seguir haciendo pruebas.





A continuación comprobaremos que efectivamente el pago se ha realizado con éxito, utilizando la misma página de pruebas e introduciendo el ID del comercio utilizado anteriormente.

Actividades

Navegador web Firefox

mar 17:53

Sistema de Pago con tarjeta - Mozilla Firefox

Curso: SISTEMAS INFI xMicrosoft Word - sl2-pr xPráctica 1: Arquitectura xSistema de Pago con tar xPráctica 1: Arquitectura xNueva pestaña x

10.1.7.4:8080/P1/testbd.jsp

# 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

Pagar

---

## Consulta de pagos

Id Comercio: 10

GetPagos

---

## Borrado de pagos

Id Comercio:

DelPagos

---

Prácticas de Sistemas Informáticos II

Actividades

Navegador web Firefox

mar 17:53

Sistema de Pago con tarjeta - Mozilla Firefox

Curso: SISTEMAS INFI xMicrosoft Word - sl2-pr xPráctica 1: Arquitectura xSistema de Pago con tar xPráctica 1: Arquitectura xNueva pestaña x

10.1.7.4:8080/P1/getpagos

# Pago con tarjeta

Lista de pagos del comercio 10

idTransaccion	Importe	codRespuesta	idAutorizacion
1	100.0	000	1

Volver al comercio

---

Prácticas de Sistemas Informáticos II

Finalmente podremos borrar el pago realizado desde la misma página de pruebas, en el apartado 'Borrado de pagos' y escribiendo el ID del comercio especificado en la transacción.

Actividades

Navegador web Firefox

mar 17:53

Sistema de Pago con tarjeta - Mozilla Firefox

Curso: SISTEMAS INFI X Microsoft Word - sl2-pr X Práctica 1: Arquitectura X Sistema de Pago con tar X Práctica 1: Arquitectura X Nueva pestaña X +

10.1.7.4:8080/P1/testbd.jsp

## 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

Pagar

### Consulta de pagos

Id Comercio:

GetPagos

### Borrado de pagos

Id Comercio:

10

DelPagos

Prácticas de Sistemas Informáticos II

Actividades

Navegador web Firefox

mar 17:53

Sistema de Pago con tarjeta - Mozilla Firefox

Curso: SISTEMAS INFI X Microsoft Word - sl2-pr X Práctica 1: Arquitectura X Sistema de Pago con tar X Práctica 1: Arquitectura X Nueva pestaña X +

10.1.7.4:8080/P1/delpagos

## Pago con tarjeta

Se han borrado 1 pagos correctamente para el comercio 10

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

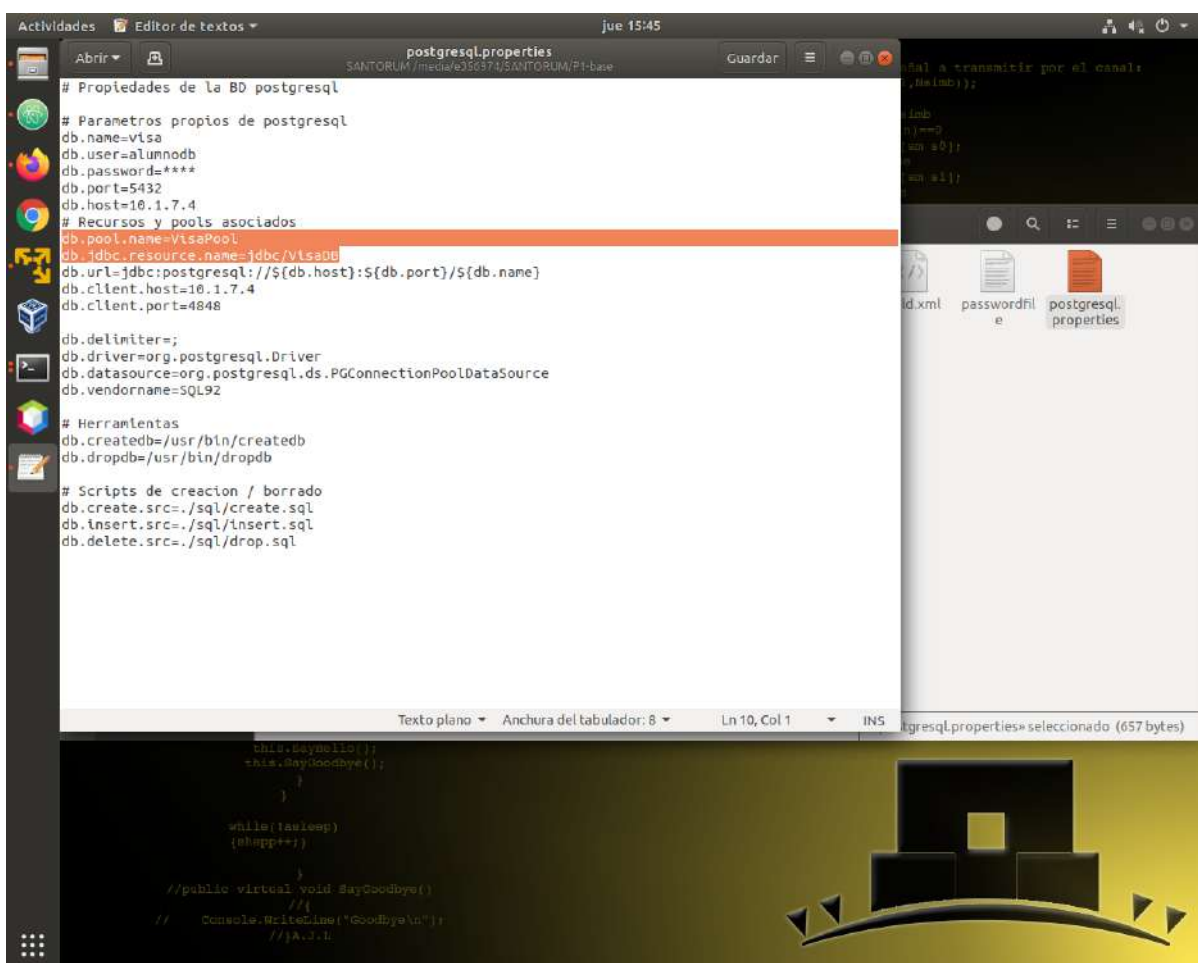
### 3 Ejercicio 3

**Enunciado:** 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.

#### Respuesta a la cuestión:

Examinando el fichero `postgresql.properties` es fácil obtener el nombre del recurso JDBC del `DataSource` y el nombre del `pool`.

- Nombre JDBC `DataSource`: `jdbc/visaDB`
- Nombre `pool`: `VisaPool`

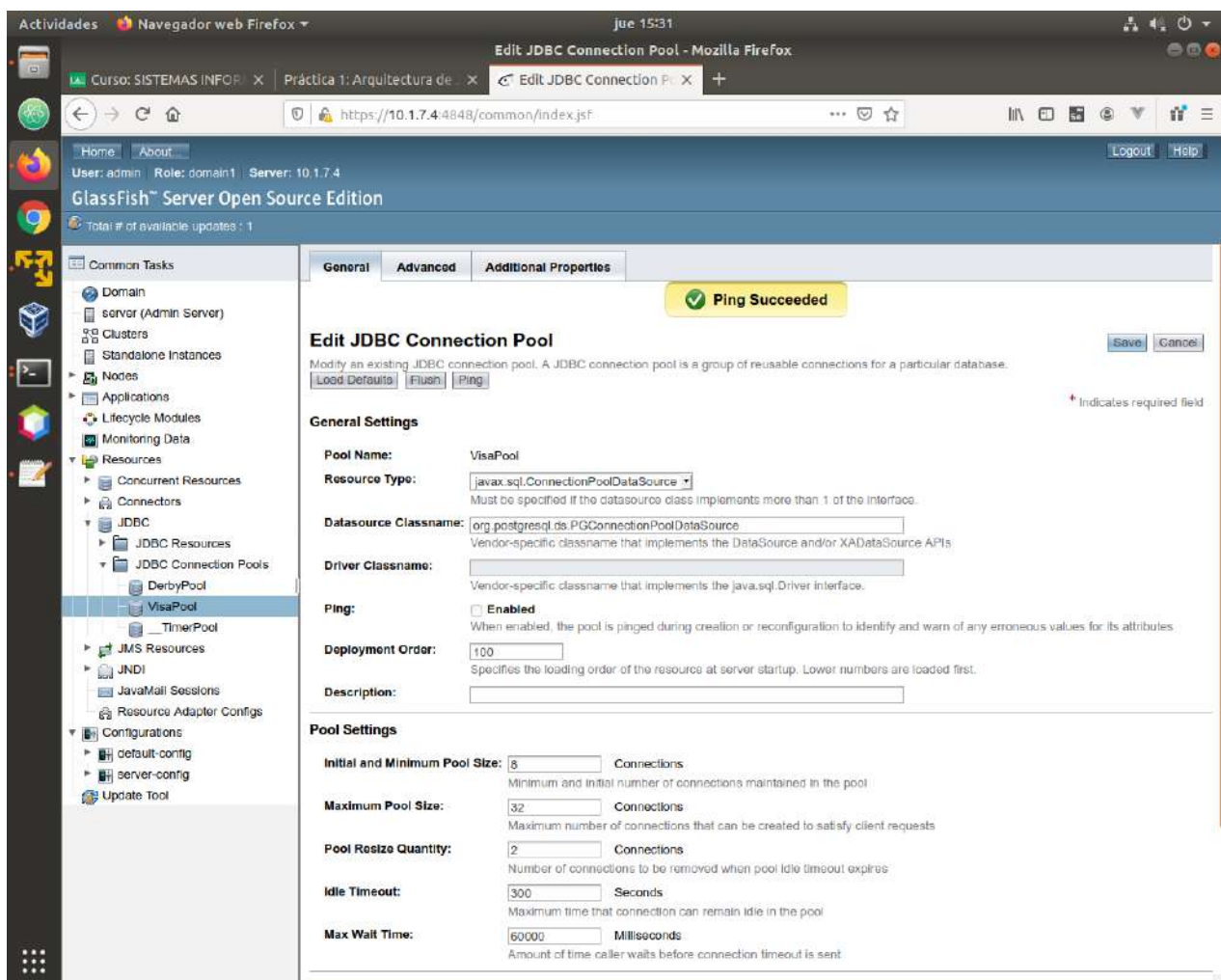


Con esta información podremos acceder a la Consola de Administración y comprobar el estado de los recursos JDBC y `pool` de conexiones.

Para ser más exactos, deberemos ir a la ruta `http://10.1.7.4:4848/` y allí podremos ver los recursos solicitados en **Common Tasks** → **Resources** → **JDBC** → **JDBC Connection Pools** → **VisaPool**.

En esta ruta, aparte de ver toda la configuración de los recursos solicitados, también podemos realizar un `Ping JDBC` a la base de datos.

Todo esto lo podemos ver en la siguiente imagen aportada.



En esta página es fácil ver los valores de las variables solicitadas.

- **Initial and Minimum Pool Size:** 8 conexiones
- **Maximum Pool Size:** 32 conexiones
- **Pool Resize Quantity:** 2 conexiones
- **Idle Timeout:** 300 segundos
- **Max Wait Time:** 60000 milisegundos

La constante **Initial and Minimum Pool Size** indica el número mínimo e inicial de hilos de la *pool*. Dependiendo del número de solicitudes de conexión este número debería variar si buscamos tener un buen rendimiento. Si nuestra aplicación tiene un tráfico constante y de un tamaño alto, este número debería ser alto, evitando levantar continuamente nuevas conexiones. Por el contrario, si nuestra aplicación tiene un tráfico relativamente bajo, deberíamos configurar este valor con un número menor para no sobrecargar el sistema de hilos en paralelo levantados.

La misma filosofía se puede utilizar con la constante **Maximum Pool Size**, que es la cantidad máxima de conexiones creables para atender a clientes. Si tenemos mucho tráfico de clientes, este número debería ser alto; y bajo si nuestro número de clientes habituales es menor.

También, con un pensamiento parecido, podemos analizar la constante **Pool Resize Quantity**, que es la cantidad de hilos que se crean (caso a) o se destruyen (caso b) de la *pool* de hilos cuando se requieren más conexiones (caso a) o cuando hay varios hilos inactivos durante mucho tiempo (caso b).

Finalmente, las constantes **Idle Timeout** y **Max Wait Time** se utilizan para establecer el tiempo máximo de espera por un hilo a ser solicitado y el tiempo máximo que el servidor espera a que el cliente actúe respectivamente. Estos dos valores tienen que ser medianamente analizados ya que valores bajos provocarán que los hilos se destruyan muy rápido si **Idle Timeout** es bajo o que el servidor no le proporcione tiempo suficiente al cliente para contestar si el valor **Max Wait Time** es bajo. Análogamente, si los valores son altos, provocarán largas esperas innecesarias, reduciendo el rendimiento y aumentando la sobrecarga del servidor.



## 4 Ejercicio 4

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

- Consulta de si una tarjeta es válida.
- Ejecución del pago.

Incluya en la memoria de prácticas dichas consultas.

**Respuesta a la cuestión:**

Se pueden encontrar en el fichero **VisaDAO.java**, situado en el directorio:

P1-base/src/ssii2/visa/dao, en las siguientes líneas:

- Consulta de si una tarjeta es válida: método **getQryCompruebaTarjeta** entre las líneas 88 y 96.
- Consulta de ejecución del pago: método **getQryInsertPago** entre las líneas 101 y 113.

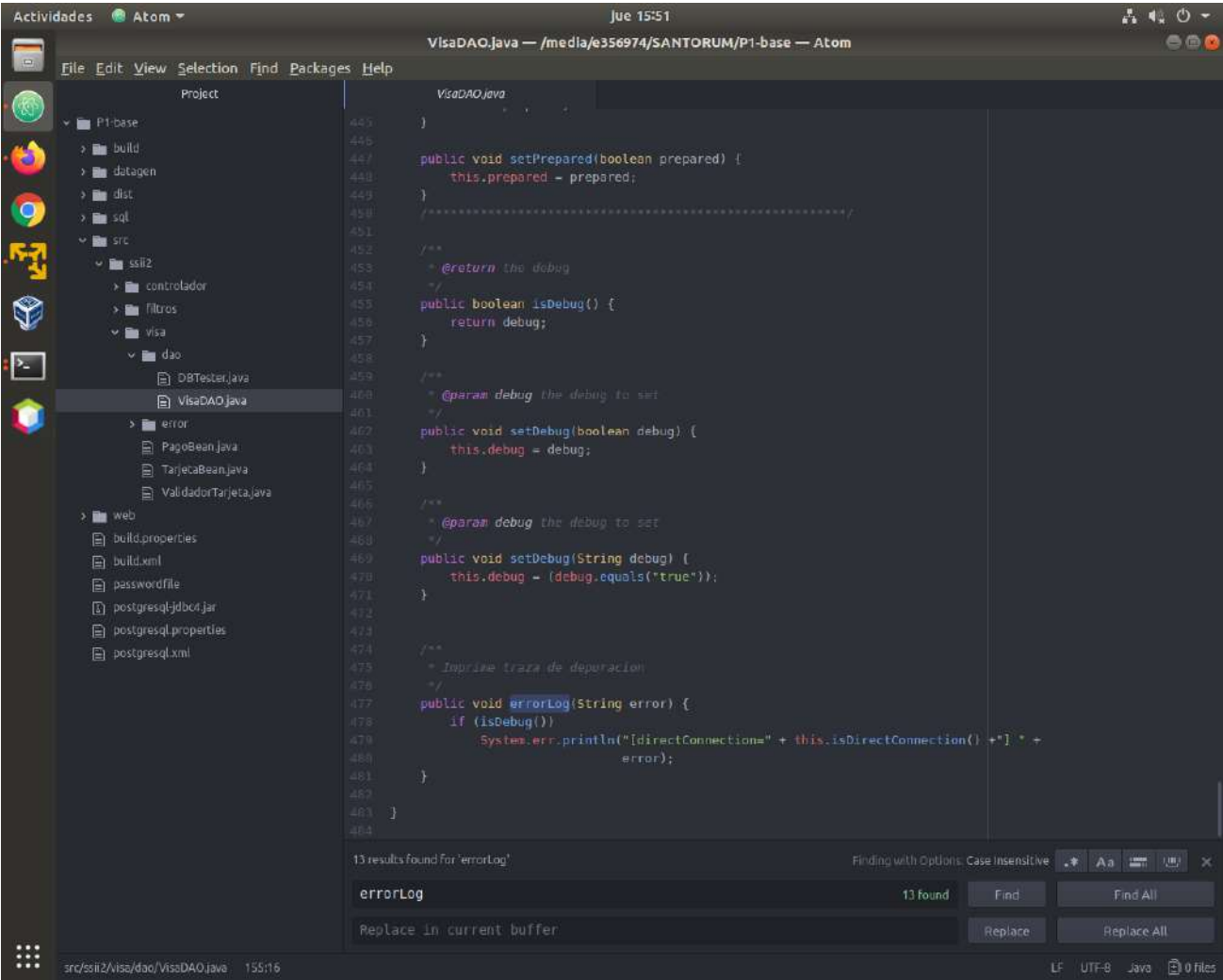
```
77  /**
78   * Constructor de la clase
79   */
80  public VisaDAO() {}
81      return;
82  }
83
84
85  /**
86   * getQryCompruebaTarjeta
87   */
88  String getQryCompruebaTarjeta(TarjetaBean tarjeta) {
89      String qry = "select * from tarjeta "
90          + "where numeroTarjeta=" + tarjeta.getNumero()
91          + " and titular=" + tarjeta.getTitular()
92          + " and validaDesde=" + tarjeta.getFechaEmision()
93          + " and validaHasta=" + tarjeta.getFechaCaducidad()
94          + " and codigoVerificacion=" + tarjeta.getCodigoVerificacion() + """;
95
96      return qry;
97  }
98
99  /**
100   * getQryInsertPago
101   */
102  String getQryInsertPago(PagoBean pago) {
103      String qry = "insert into pago("
104          + "idTransaccion,"
105          + "importe,idComercio,"
106          + "numeroTarjeta)"
107          + " values ("
108          + "" + pago.getIdTransaccion() + ","
109          + pago.getImporte() + ","
110          + "" + pago.getIdComercio() + ","
111          + "" + pago.getTarjeta().getNumero() + ""
112          + ")";
113
114      return qry;
115  }
116
117  /**
118   * getQryBuscaPagoTransaccion()
119   */
120  String getQryBuscaPagoTransaccion(PagoBean pago) {
121      String qry = "select idAutorizacion, codRespuesta "
122          + " from pago "
123          + " where idTransaccion = " + pago.getIdTransaccion()
124          + " and idComercio = " + pago.getIdComercio() + """;
```

## 5 Ejercicio 5

**Enunciado:** 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.

**Respuesta a la cuestión:**

Entrando en el fichero **VisaDAO.java**, situado en el directorio ya mencionado anteriormente, podemos buscar el método **errorLog**, que se usa 12 veces en el código de ese fichero para realizar acciones de *debugging* y mostrar trazas de ejecución más significativas que las que habitualmente salen.

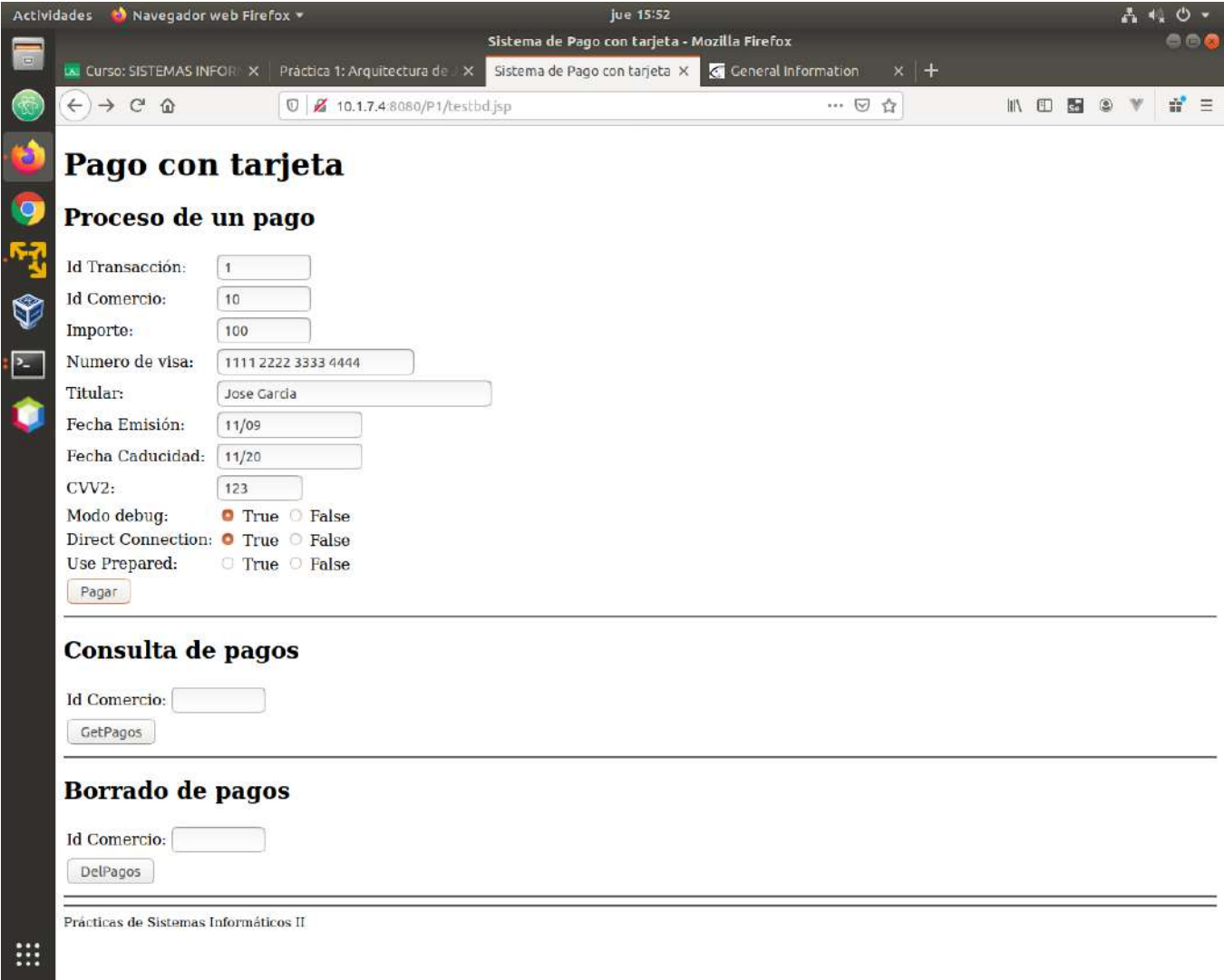


Las partes en donde se utiliza este método son en otros métodos que acceden a la base de datos con el objetivo de hacer una consulta y/o modificación. Al tratarse de procedimientos delicados, es necesario preservar en el *log* trazas exhaustivas por si se da el caso en que algo falla y poder actuar y arreglarlo.

Los métodos que se ayudan de errorLog son, principalmente, el método de **comprobación de tarjeta válida**, el método de **insercción de pago**, el procedimiento de **consulta de pagos** y el método de **eliminación de pagos**.

A continuación realizaremos un pago y observaremos qué trazas obtenemos en el log del servidor de aplicaciones.

Utilizaremos para ello los datos obtenidos en el ejercicio 1 y con la opción de **debug activada**.



Pulsando en 'Pagar' deberíamos obtener la siguiente salida, un pago exitoso.



Si comprobamos en el servidor de aplicaciones el *log* podemos ver qué ha ocurrido por detrás.

Actividades

Navegador web Firefox

Jue 15:53

Log Viewer - Mozilla Firefox

https://10.1.7.4:4848/common/logViewer/logViewer.jsf?instanceName=server&logLevel=INFO&viewResults=true

Log Level: INFO 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: server

Log File: server.log

Log Viewer Results (40)

Records before 328 Log File Record Numbers 328 through 367 Records after 367

Record Number	Log Level	Message	Logger	Timestamp	Name-Value Pairs
367	INFO	WebModule[null] ServletContext.log(): [INFO] Acceso correcto: testId.jsp (details)	javax.enterprise.web	Feb 13, 2020 06:32:46.352	(levelValue=800, timeMillis=1581605586352)
366	INFO	WebModule[null] ServletContext.log(): [INFO] Acceso correcto: testId.jsp (details)	javax.enterprise.web	Feb 13, 2020 06:32:45.458	(levelValue=800, timeMillis=1581605585458)
365	SEVERE	[directConnection=true] select idAutorizacion, codRespuesta from pago where idTransaction = '1'... (details)		Feb 13, 2020 06:32:35.327	(levelValue=1000, timeMillis=1581605553327)
364	SEVERE	[directConnection=true] Insert into pago(idTransaction, importe, idComercio, numeroTarjeta) values ('1', 3333.4444 and titular=... (details)		Feb 13, 2020 06:32:35.325	(levelValue=1000, timeMillis=1581605553325)
363	SEVERE	[directConnection=true] select * from tarjeta where numeroTarjeta='1111 2222 3333 4444' and titular=... (details)		Feb 13, 2020 06:32:35.323	(levelValue=1000, timeMillis=1581605553323)
362	INFO	WebModule[null] ServletContext.log(): [INFO] Acceso correcto: procesapago (details)	javax.enterprise.web	Feb 13, 2020 06:32:35.317	(levelValue=800, timeMillis=1581605553317)
361	INFO	WebModule[null] ServletContext.log(): [INFO] Acceso correcto: testId.jsp (details)	javax.enterprise.web	Feb 13, 2020 06:32:10.814	(levelValue=800, timeMillis=1581605530814)
360	INFO	Admin Console: Initializing Session Attributes... (details)	org.glassfish.admin.ui	Feb 13, 2020 06:28:31.461	(levelValue=800, timeMillis=1581604111461)
359	INFO	Redirecting to /index.jsf (details)	org.glassfish.admin.ui	Feb 13, 2020 06:28:31.362	(levelValue=800, timeMillis=1581604111362)
358	INFO	Listening to REST requests at context: /managment/domain/ (details)	javax.enterprise.admin.rest	Feb 13, 2020 06:28:31.349	(levelValue=800, timeMillis=1581604111349)
357	SEVERE	The SSL certificate has expired: ([ Version: V3 Subject: CN=Entrust.net Secure Server Certific... (details)	javax.enterprise.system.security.ssl	Feb 13, 2020 06:28:29.987	(levelValue=1000, timeMillis=1581604109987)
356	SEVERE	The SSL certificate has expired: ([ Version: V3 Subject: CN=Entrust.net Certification Authority... (details)	javax.enterprise.system.security.ssl	Feb 13, 2020 06:28:29.986	(levelValue=1000, timeMillis=1581604109986)
355	SEVERE	The SSL certificate has expired: ([ Version: V1 Subject: EMAILADDRESS=info@valicert.com, CN=Hit... (details)	javax.enterprise.system.security.ssl	Feb 13, 2020 06:28:29.986	(levelValue=1000, timeMillis=1581604109986)
354	SEVERE	The SSL certificate has expired: ([ Version: V3 Subject: CN=Class 2 Primary CA, O=Certplus, C=F... (details)	javax.enterprise.system.security.ssl	Feb 13, 2020 06:28:29.985	(levelValue=1000, timeMillis=1581604109985)
353	SEVERE	The SSL certificate has expired: ([ Version: V1 Subject: CN=GTE CyberTrust Global Root, OU=GTE... (details)	javax.enterprise.system.security.ssl	Feb 13, 2020 06:28:29.985	(levelValue=1000, timeMillis=1581604109985)
352	SEVERE	The SSL certificate has expired: ([ Version: V3 Subject: CN=UTN - DATACorp SGC, OU=http://www.u... (details)	javax.enterprise.system.security.ssl	Feb 13, 2020 06:28:29.985	(levelValue=1000, timeMillis=1581604109985)
351	SEVERE	The SSL certificate has expired: ([ Version: V3 Subject: CN=Class 3P Primary CA, O=Certplus, C=... (details)	javax.enterprise.system.security.ssl	Feb 13, 2020 06:28:29.984	(levelValue=1000, timeMillis=1581604109984)
350	SEVERE	The SSL certificate has expired: ([ Version: V3 Subject: CN=UTN-USERFirst-Object, OU=http://www... (details)	javax.enterprise.system.security.ssl	Feb 13, 2020 06:28:29.984	(levelValue=1000, timeMillis=1581604109984)
349	SEVERE	The SSL certificate has expired: ([ Version: V3 Subject: CN=Deutsche Telekom Root CA 2, OU=T-Te... (details)	javax.enterprise.system.security.ssl	Feb 13, 2020 06:28:29.983	(levelValue=1000, timeMillis=1581604109983)
348	SEVERE	The SSL certificate has expired: ([ Version: V1 Subject:		Feb 13, 2020	(levelValue=1000,

Primero se consulta que la tarjeta sea la correcta:

Actividades

Navegador web Firefox

Jue 15:53

Log Entry Detail - Mozilla Firefox

https://10.1.7.4:4848/common/logViewer/logEntryDetail.jsf?instanceName=server&logLevel=SEVERE&logFile=server.log&recNumber=363

Log Entry Detail

Timestamp Feb 13, 2020 06:32:35.323

Log Level SEVERE

Logger

Name-Value Pairs (levelValue=1000, timeMillis=1581605553320)

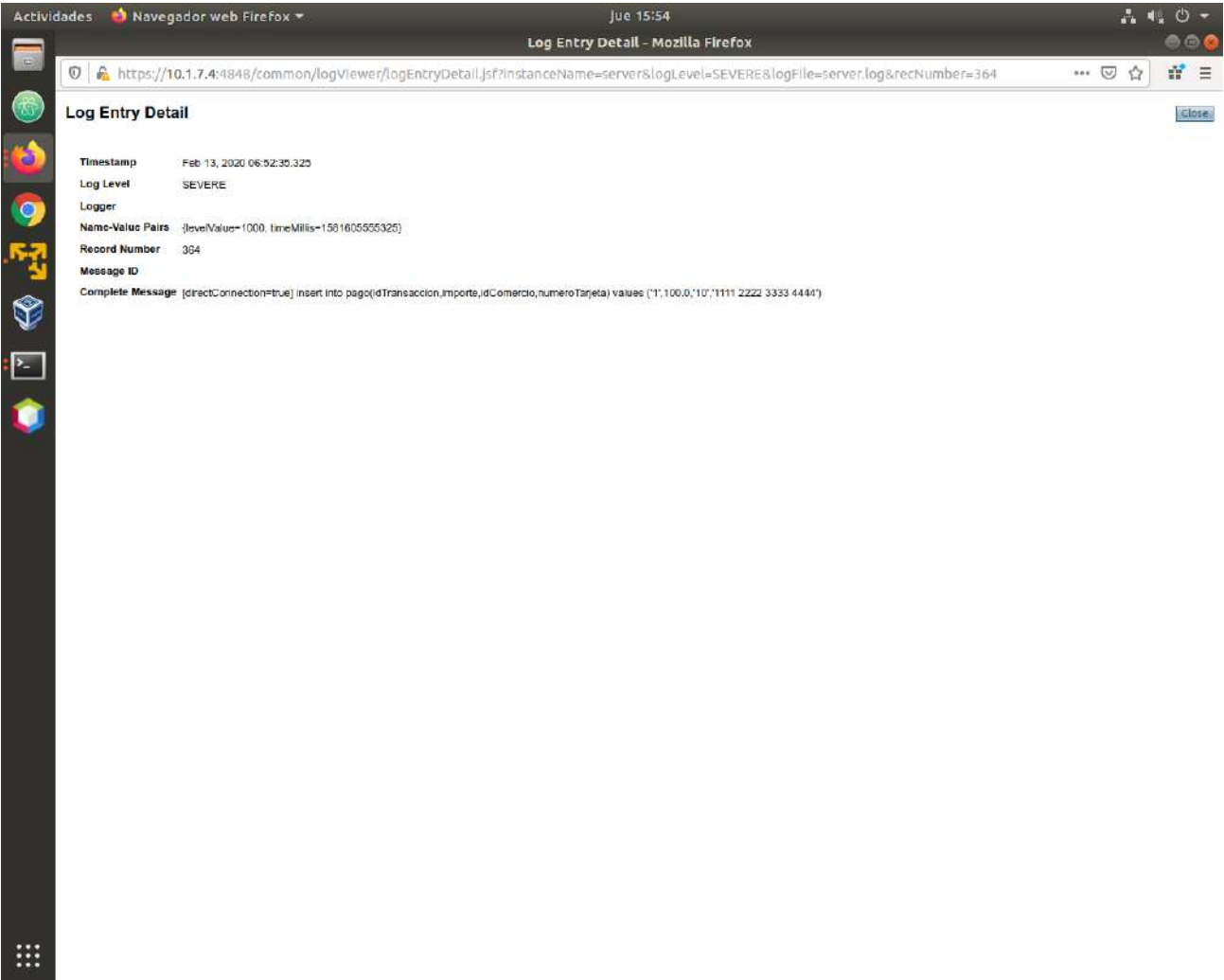
Record Number 363

Message ID

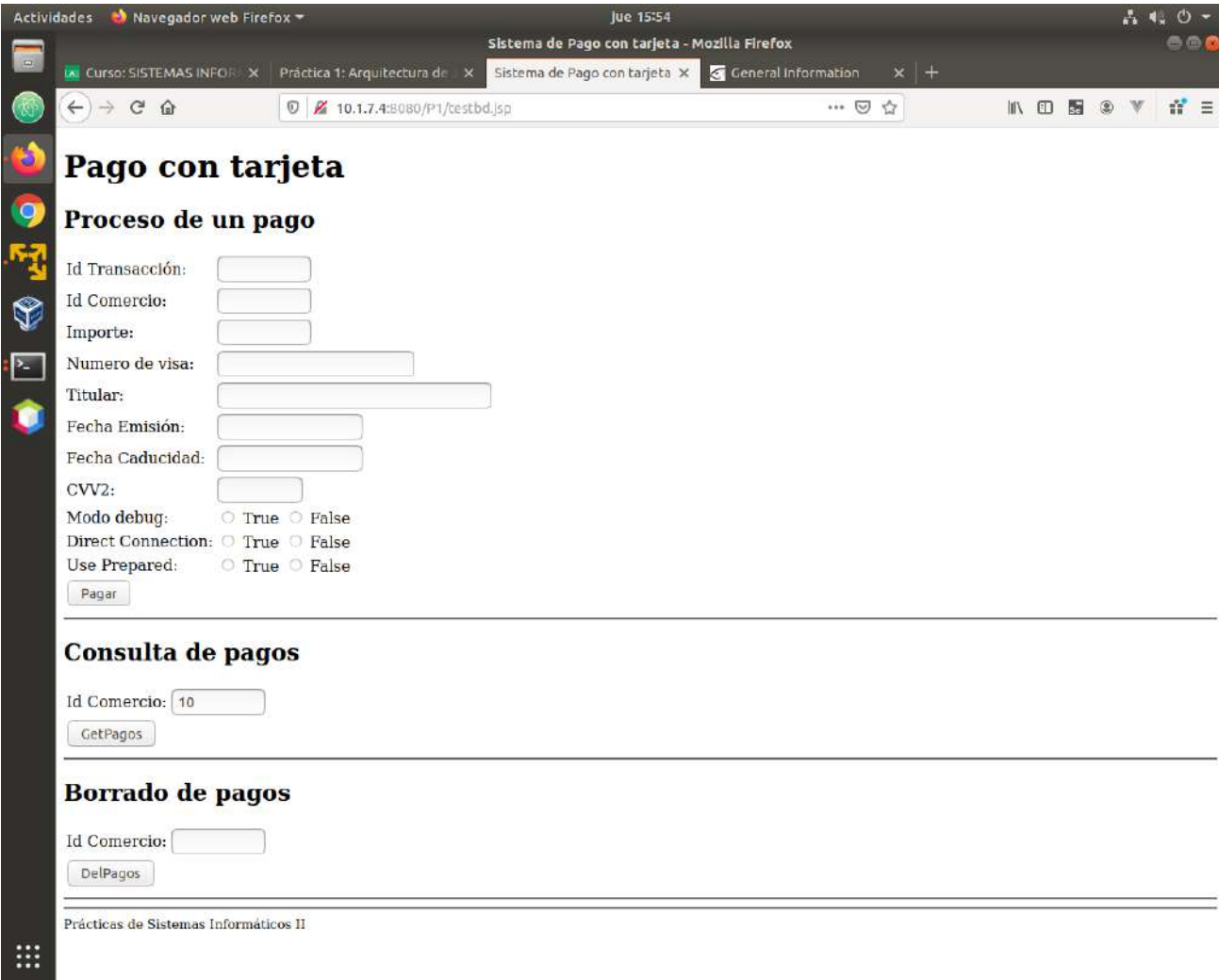
Complete Message [directConnection=true] select \* from tarjeta where numeroTarjeta='1111 2222 3333 4444' and titular='Jose Garcia' and validaDesde='11/06' and validaHasta='11/20' and codigoVerificacion='123'



Y después, si la tarjeta es correcta, se inserta el pago con la información aportada:

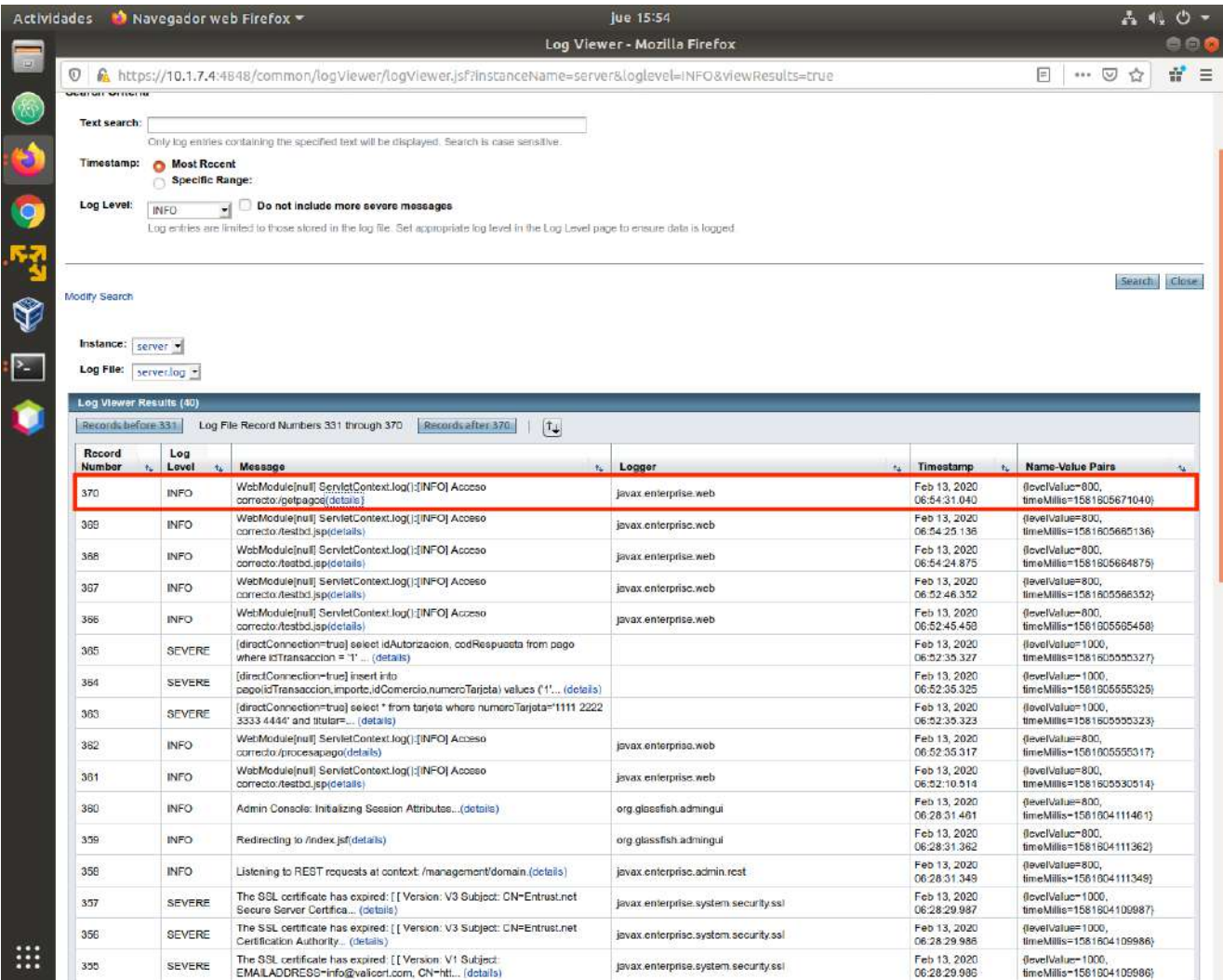


Ahora vamos a comprobar el pago en la página extendida con el ID del comercio anteriormente suministrado.

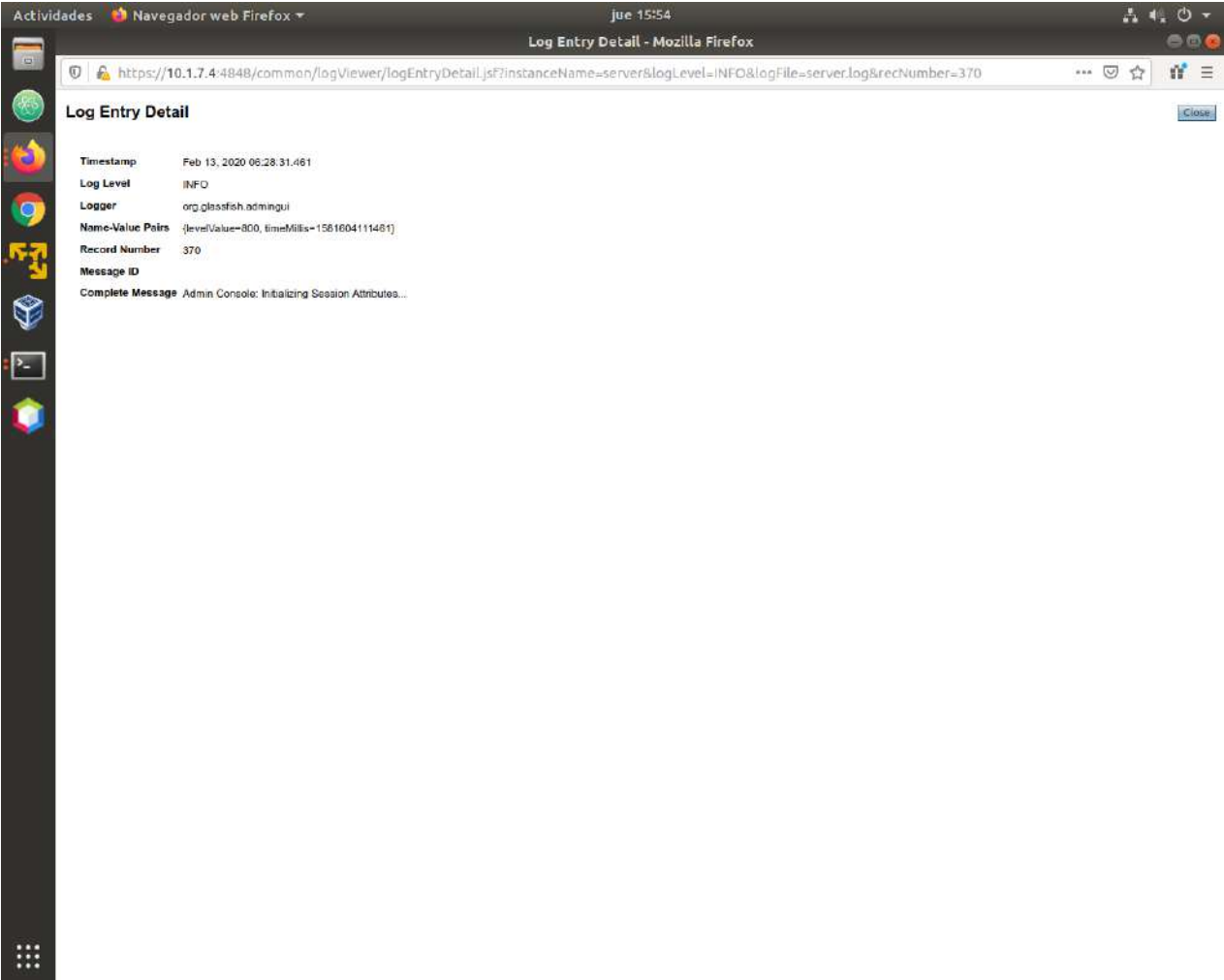




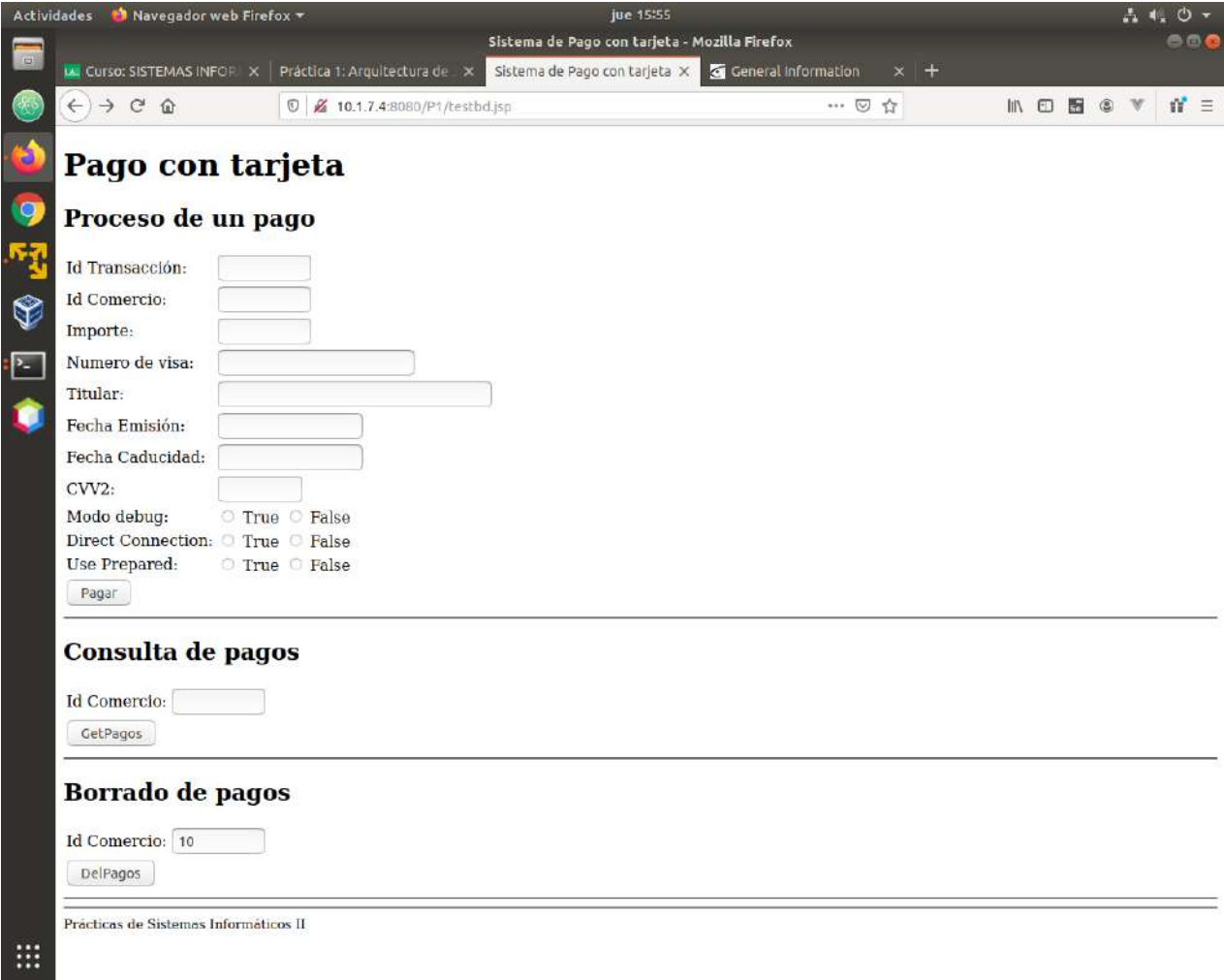
Y vemos en el *log* qué se ha ejecutado:



Como podemos ver se ha realizado la consulta **getPagos** exitosamente, devolviendo los pagos realizados.



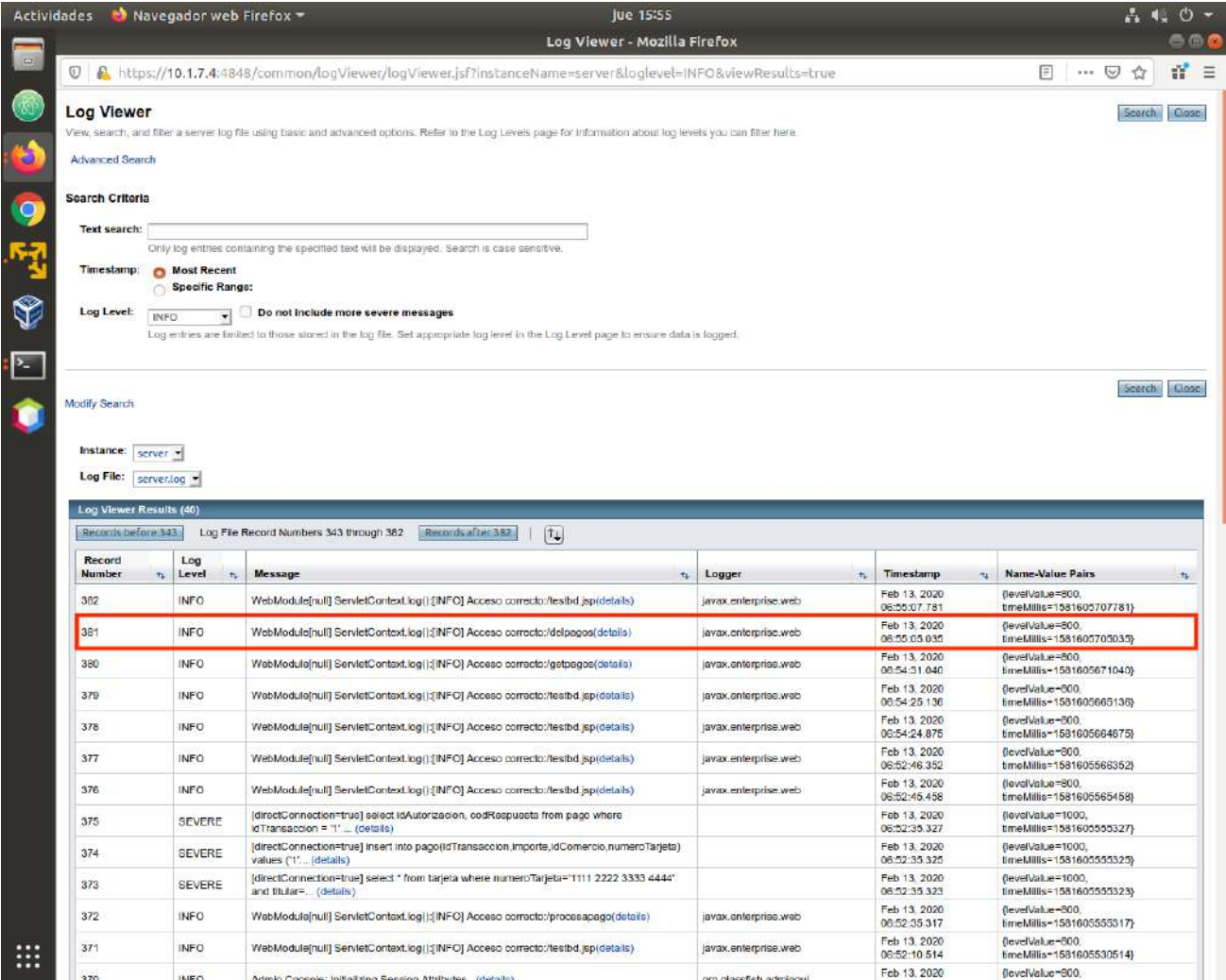
Finalmente, repetiremos este procedimiento para eliminar los pagos del comercio con el ID utilizado.



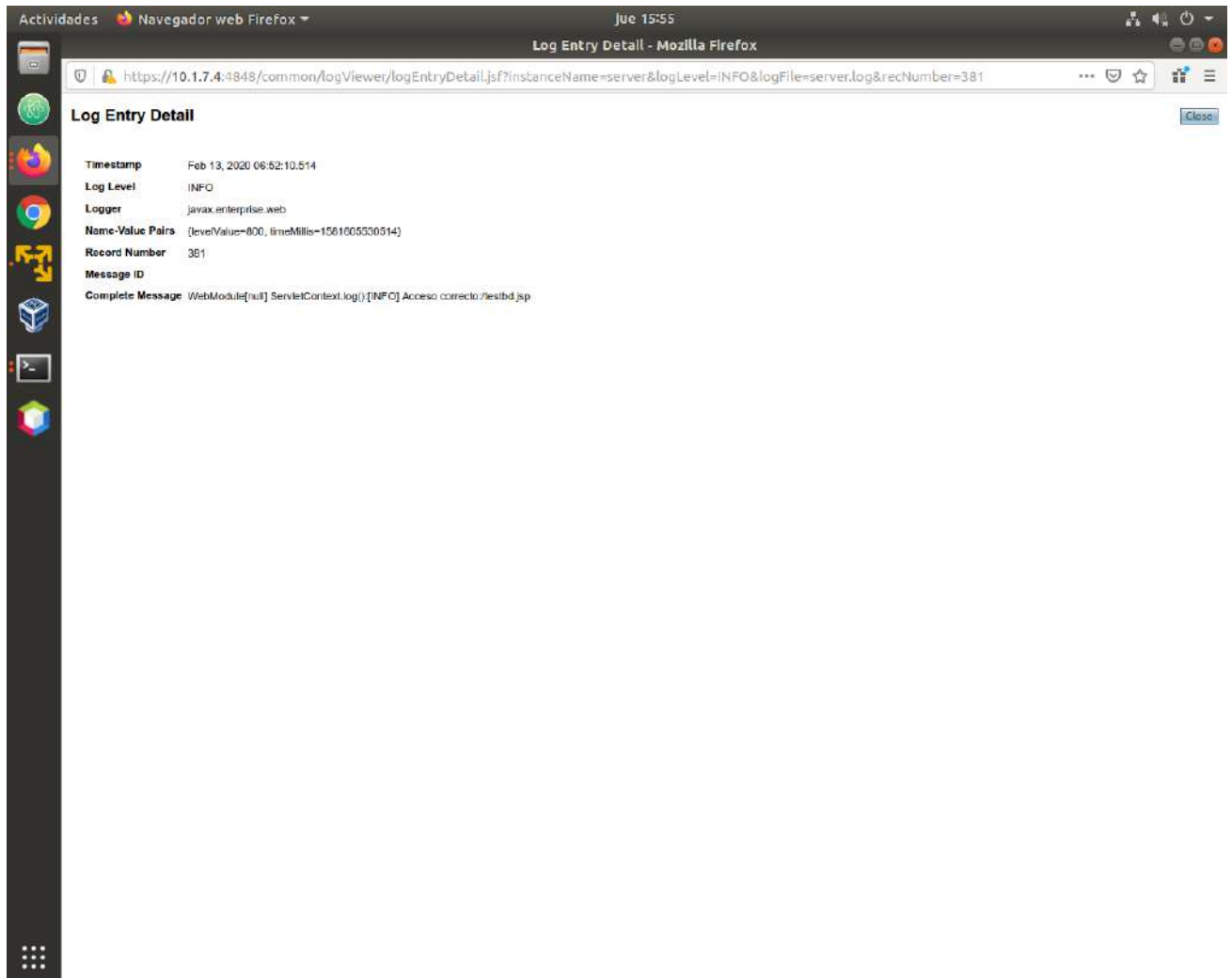
En la página extendida obtenemos un mensaje de eliminación exitosa.



Y en el *log* del servidor de aplicaciones también podemos ver que se ha ejecutado satisfactoriamente la consulta **delPagos**:







## 6 Ejercicio 6

### Enunciado:

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()

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

- isDirectConnection() / setDirectConnection()

Para ello, implemente estos métodos también en la clase hija. Es decir, haga un override de Java, implementando estos métodos en VisaDAOWS mediante invocaciones a la clase padre (super). En ningún caso se debe añadir ni modificar nada de la clase DBTester.

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

- Con identificador de autorización y código de respuesta correcto en caso de haberse realizado.
- Con null en caso de no haberse podido realizar.

Incluye en la memoria cada fragmento de código donde se han ido añadiendo las modificaciones requeridas.

Por último, conteste a la siguiente pregunta:

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

### Respuesta a la cuestión:

Para comenzar, añadimos los *imports* especificados en el enunciado en el fichero `VisaDAOWS.java`. A continuación cambiamos el nombre de la clase por el nuevo: **VisaDAOWS**

```
/* Imports para transformar la clase en un webService */
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;

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

    /* (...) */

    /**
     * Constructor de la clase
     */
    public VisaDAOWS() {
        return;
    }
}
```

Después, tenemos que añadir las anotaciones que indican que implementamos un nuevo servicio web y sus métodos, así como cambiar los retornos de la función `realizaPago`, ya que ahora devuelve un `PagoBean` en lugar de un `boolean/null`.

```
@WebMethod(operationName = "compruebaTarjeta")
public boolean compruebaTarjeta(@WebParam(name = "tarjeta") TarjetaBean tarjeta) {
    // (...)
}

@WebMethod(operationName = "realizaPago")
public synchronized PagoBean realizaPago(@WebParam(name = "pago") PagoBean pago) {
    // (...)
    ret = null;
    if (!pstmt.execute()
        && pstmt.getUpdateCount() == 1) {
        ret = pago;
    }
    // Cambiamos todos los ret = false por ret = null y ret = true por ret = pago.
    // (...)
}

@WebMethod(operationName = "isPrepared")
public boolean isPrepared() {
    // (...)
}

@WebMethod(operationName = "setPrepared")
public void setPrepared(boolean prepared) {
    // (...)
}

@WebMethod(operationName = "isDebug")
public boolean isDebug() {
    // (...)
}

@WebMethod(operationName = "setDebug")
public void setDebug(boolean debug) {
    // (...)
}

@WebMethod(exclude = true)
public void setDebug(String debug) {
    // (...)
}

@WebMethod(operationName = "isDirectConnection")
@Override
public boolean isDirectConnection() {
    return super.isDirectConnection();
}

@WebMethod(operationName = "setDirectConnection")
@Override
public void setDirectConnection(@WebParam(name = "directConnection") boolean directConnection) {
    super.setDirectConnection(directConnection);
}
}
```

## 7 Ejercicio 7

**Enunciado:**

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

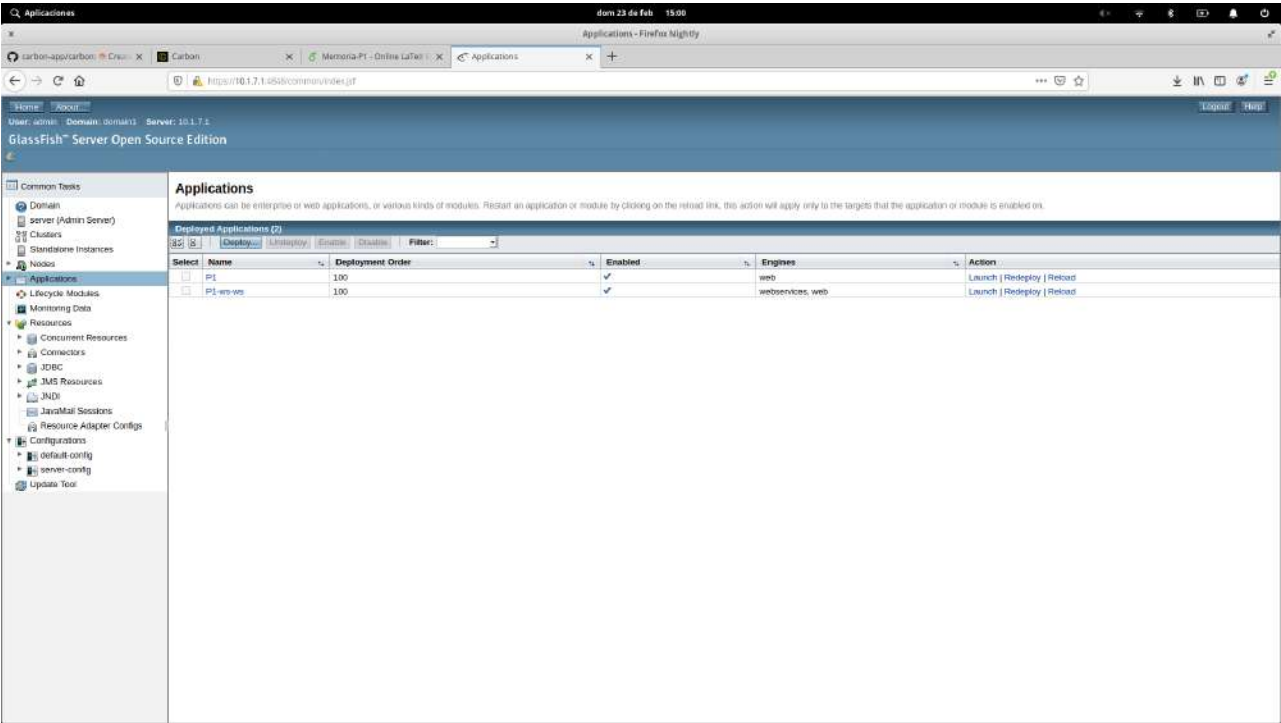
Comente en la memoria aspectos relevantes del código XML del fichero WSDL y su relación con los métodos Java del objeto del servicio, argumentos recibidos y objetos devueltos.

Conteste a las siguientes preguntas:

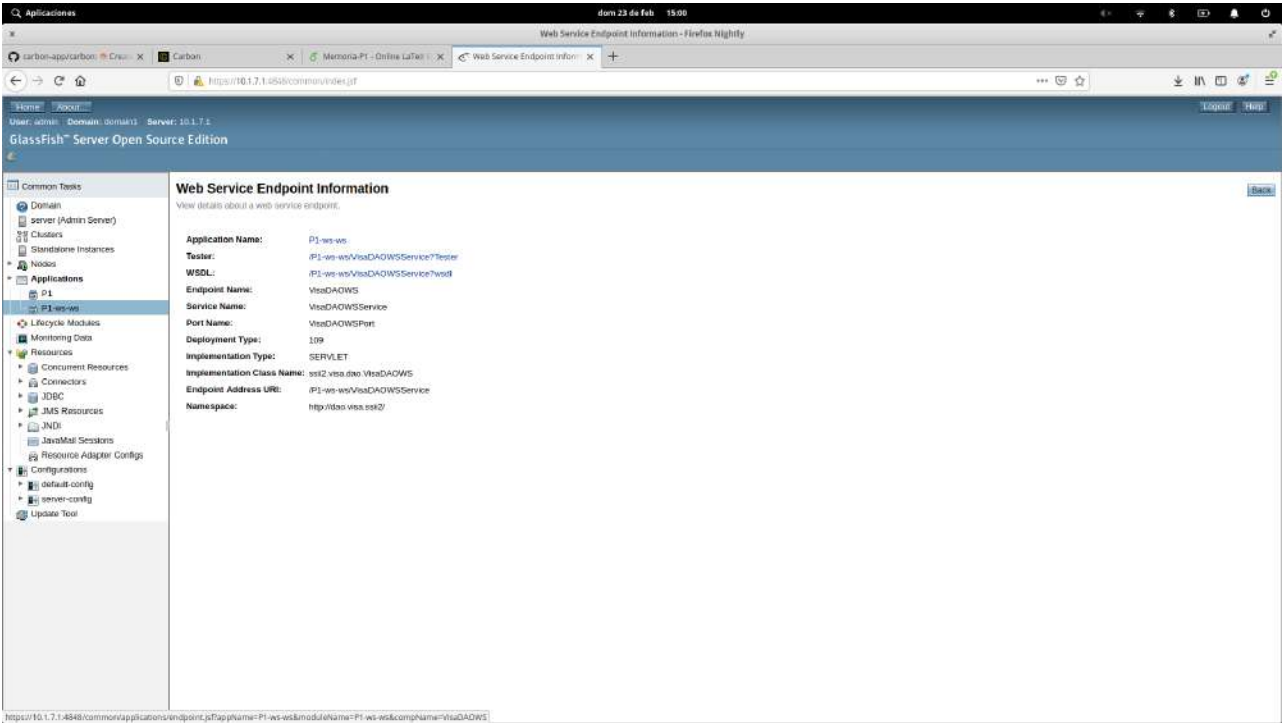
- ¿En qué fichero están definidos los tipos de datos intercambiados con el webservice?
- ¿Qué tipos de datos predefinidos se usan?
- ¿Cuáles son los tipos de datos que se definen?
- ¿Qué etiqueta está asociada a los métodos invocados en el webservice?
- ¿Qué etiqueta describe los mensajes intercambiados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

**Respuesta a la cuestión:**

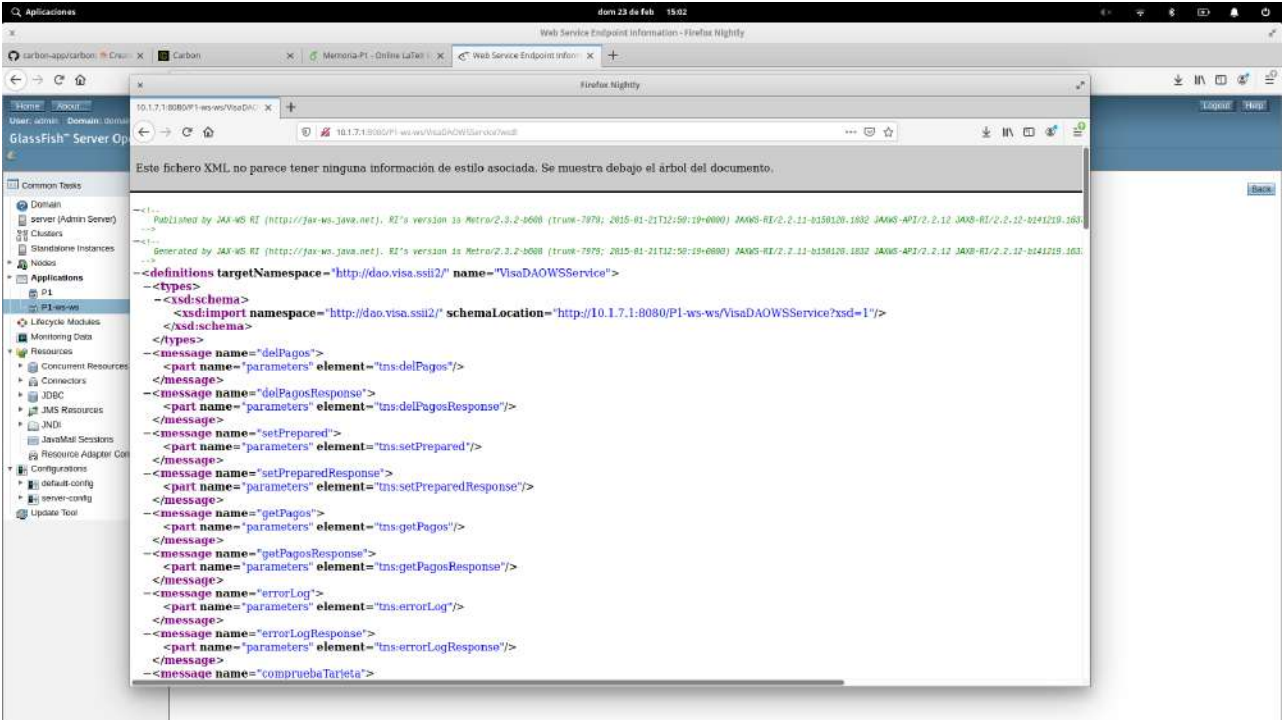
Desplegamos el servicio utilizando los comandos aportados en el fichero build.xml. Ahora accedemos a la página de administración en **http://10.1.7.1:4848** y en 'Applications' podemos encontrar **P1-ws-ws**:



Pinchando en **P1-ws-ws** y en 'View Endpoint' podemos ver la información del web service



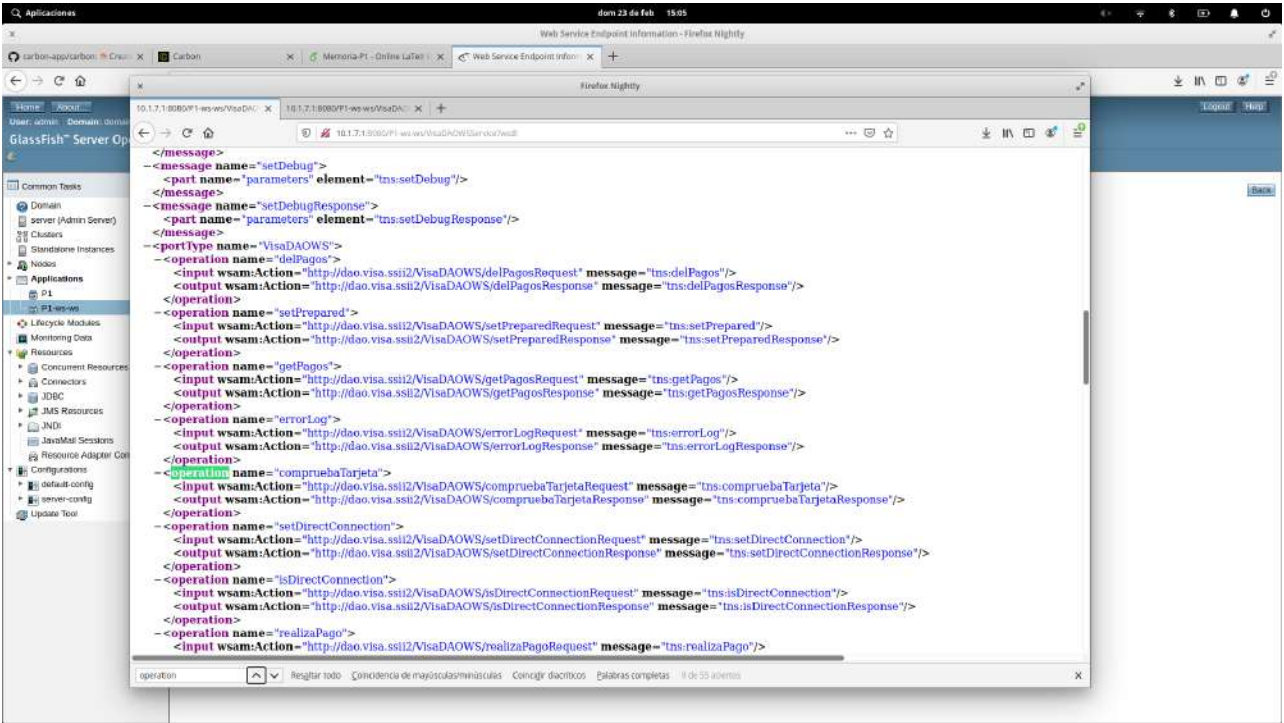
Desde aquí podemos acceder al WSDL de la aplicación pinchando en el tercer enlace que nos aparece (/P1-ws-ws/VisaDAOWSService?wsdl, recuerde que la URL contiene la dirección IP de la máquina virtual donde se encuentra el servidor de aplicaciones). Nos debería aparecer el siguiente fichero:



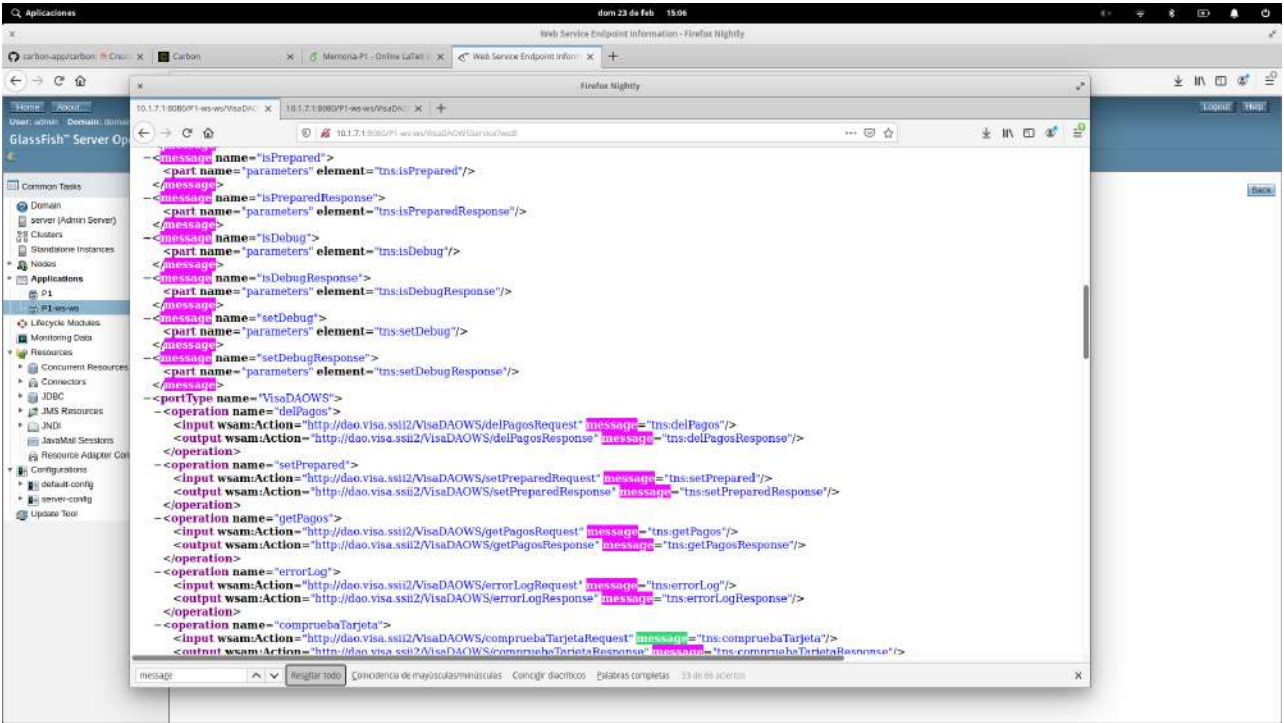
Con la información recogida en este fichero podremos contestar a las preguntas de este ejercicio. En primer lugar se nos pregunta en qué fichero están definidos los tipos de datos intercambiados con el webservice. Este fichero es el aportado en la etiqueta `xxsd:import namespace=... schemaLocation=...`, para ser más exactos, es el enlace situado en 'schemaLocation'. Los tipos de datos predefinidos que se usan se pueden observar en este último fichero, bajo la etiqueta `xs:element ... type="xs:..."`





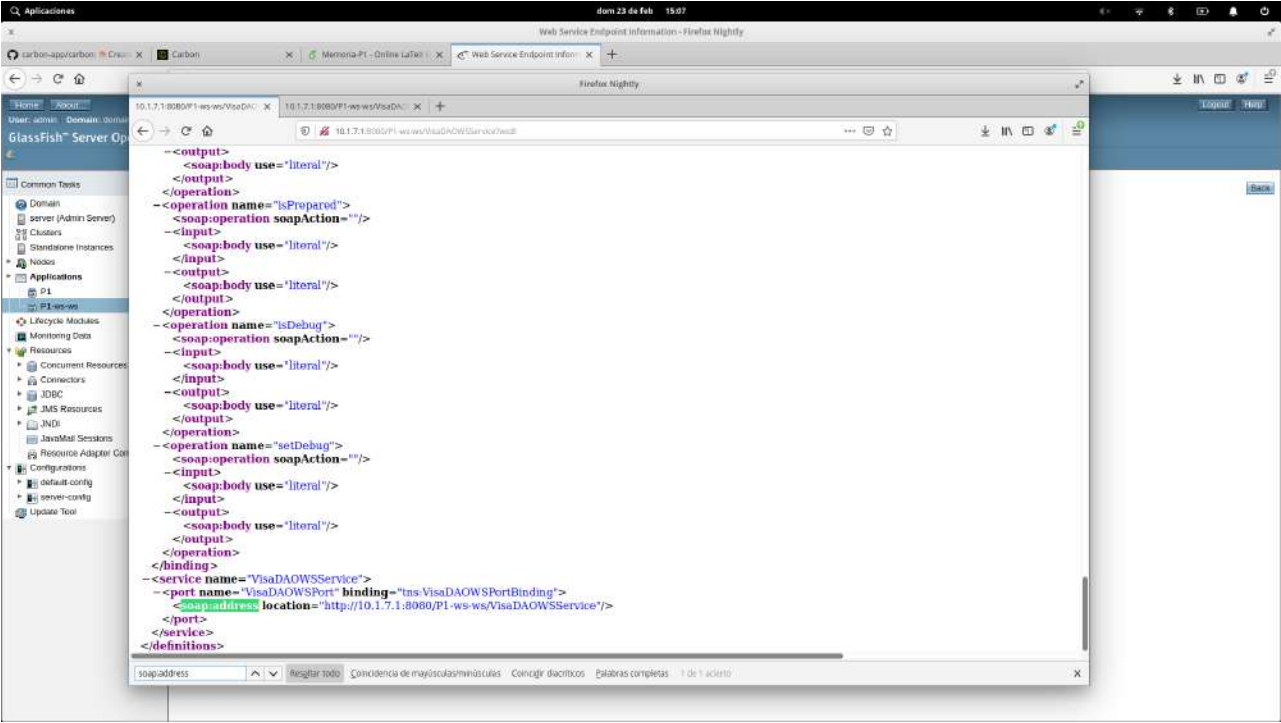
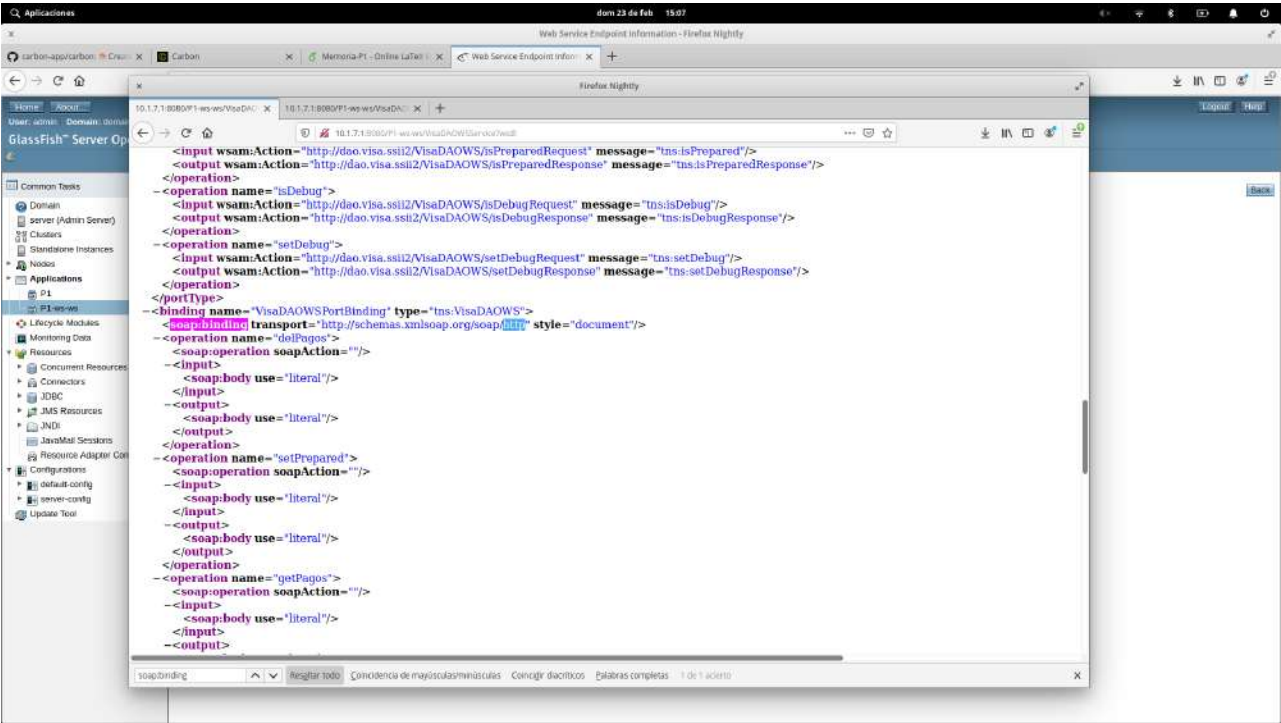


Y también se pide la etiqueta que describe los mensajes intercambiados en la invocación de los métodos del webservice. Esta es `message`:



Como podemos observar, la etiqueta `message` también sirve para especificar los tipos de parámetros de entrada y salida de las operaciones entre el cliente y el servidor de aplicaciones.

Finalmente, se nos pregunta por la etiqueta que especifica el protocolo de comunicación con el webservice, y la etiqueta que especifica la URL a la que se deberá conectar un cliente para acceder al webservice. La primera es la etiqueta `soap:binding` y la segunda es la etiqueta `soap:address`:





## 8 Ejercicio 8

### Enunciado:

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

### Respuesta a la cuestión:

Como se nos ha explicado, realizamos los cambios necesarios en el fichero **ProcesaPago.java**: añadimos los *imports* especificados, realizamos la instanciación de la clase remota en dos pasos y tenemos en cuenta las posibles nuevas excepciones y que `realizaPago` ahora devuelve un `PagoBean` en lugar de un boolean/null.

```
// ProcesaPago.java
// (...)
// import ssii2.visa.dao.VisaDAO;

/* Imports para instanciar un stub */
import ssii2.visa.VisaDAOWSService;
import ssii2.visa.VisaDAOWS;
import javax.xml.ws.WebServiceRef;

// (...)
VisaDAOWSService service = null;
VisaDAOWS dao = null;

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

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

## 9 Ejercicio 9

### Enunciado:

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.



### Respuesta a la cuestión:

Modificamos primero el fichero **web.xml** para añadir un parámetro de inicialización tal y como se nos indica en el apéndice 15.1:

```
<!-- web.xml -->

<context-param>
  <param-name>pathVisaWS</param-name>
  <param-value>http://10.1.7.1:8080/P1-ws-ws/VisaDAOWSService</param-value>
</context-param>
```

Finalmente, modificamos el fichero **ProcesaPago.java** para actualizar la llamada al servicio para que la ruta al servicio remoto se obtenga del fichero de configuración **web.xml**

```
// ProcesaPago.java

// (...)
/* Imports para instanciar un stub */
import ssii2.visa.VisaDAOWSService;
import ssii2.visa.VisaDAOWS;
import javax.xml.ws.WebServiceRef;
import javax.xml.ws.BindingProvider;

// (...)
VisaDAOWSService service = null;
VisaDAOWS dao = null;
BindingProvider bp = null;

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

  bp = (BindingProvider) dao;
  bp.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
  getServletContext().getInitParameter("pathVisaWS"));
} catch (Exception e) {
  enviaError(e, request, response);
  return;
}
```

## 10 Ejercicio 10

### Enunciado:

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

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

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

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

array de tipo `PagoBean[]` utilizando el método `toArray()` de la clase `ArrayList`. Incluye en la memoria una captura con las adaptaciones realizadas.

### Respuesta a la cuestión:

Realizamos cambios muy parecidos a los hechos en el fichero `ProcesaPago.java` en los ficheros `VisaDAOWS.java`, `DelPagos.java` y `GetPagos.java`.

Empezamos cambiando el retorno de la función `getPagos(...)` del fichero `VisaDAOWS.java`. Además añadimos los decoradores `@WebMethod` y `@WebParam` en los métodos `getPagos` y `delPagos` del mismo fichero.

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

    // PagoBean[] ret = null;
    // (...)

    // ret = new PagoBean[pagos.size()];
    // ret = pagos.toArray(ret);

    return pagos;
}

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

Por otro lado, actualizamos los *imports* en los ficheros `DelPagos.java` y `GetPagos.java`. También modificamos la ruta del servidor remoto sin cambiar la definición del servicio como en el ejercicio 9:

```
// DelPagos.java

// (...)
// imports para acceder al WS remoto
import ssii2.visa.VisaDAOWSService;
import ssii2.visa.VisaDAOWS;
import javax.xml.ws.WebServiceRef;
import javax.xml.ws.BindingProvider;

// (...)
VisaDAOWSService service = null;
VisaDAOWS dao = null;
BindingProvider bp = null;

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

    bp = (BindingProvider) dao;
    bp.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
getServletContext().getInitParameter("pathVisaWS"));
} catch (Exception e) {
    enviaError(e, request, response);
    return;
}
```

```

// GetPagos.java

// (...)
// imports para usar Arrays
import java.util.ArrayList;
import java.util.List;
// imports para acceder al WS remoto
import ssii2.visa.VisaDAOWSService;
import ssii2.visa.VisaDAOWS;
import javax.xml.ws.WebServiceRef;
import javax.xml.ws.BindingProvider;

// (...)
VisaDAOWSService service = null;
VisaDAOWS dao = null;
BindingProvider bp = null;

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

    bp = (BindingProvider) dao;
    bp.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
getServletContext().getInitParameter("pathVisaWS"));
} catch (Exception e) {
    enviaError(e, request, response);
    return;
}

// (...)
// Obtención de pagos desde el webService y su casting a ArrayList desde List
List<PagoBean> pagosRet = dao.getPagos(idComercio);
ArrayList<PagoBean> pagos = new ArrayList<PagoBean>(pagosRet);

/* Casting a PagoBean[] para la vista .jsp */
PagoBean[] pagosArr = null;
pagosArr = new PagoBean[pagos.size()];
pagosArr = pagos.toArray(pagosArr);

```

## 11 Ejercicio 11

### Enunciado:

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.

### Respuesta a la cuestión:

Se han requerido ejecutar los siguientes comandos:

Primero **ant generar-stubs**, para crear los directorios y generar el .jar con el *manifest*.

Segundo **wsimport -d build/client/WEB-INF/classes -p ssii2.visa http://10.1.7.1:8080/P1-ws-ws/VisaDAOWSService?wsdl**, que crea las clases del servicio necesarias en cliente, a partir del fichero WSDL definido por el servidor. Las clases generadas son las siguientes:

- CompruebaTarjeta.class
- ErrorLogResponse.class
- IsDirectConnection.class
- package-info.class x
- SetDebugResponse.class
- TarjetaBean.class x
- CompruebaTarjetaResponse.class
- GetPagos.class

- IsDirectConnectionResponse.class
- PagoBean.class x
- SetDirectConnection.class
- VisaDAOWS.class x
- DelPagos.class
- GetPagosResponse.class
- IsPrepared.class
- RealizaPago.class
- SetDirectConnectionResponse.class
- VisaDAOWSService.class x
- DelPagosResponse.class
- IsDebug.class
- IsPreparedResponse.class
- RealizaPagoResponse.class
- SetPrepared.class
- ErrorLog.class
- IsDebugResponse.class
- ObjectFactory.class x
- SetDebug.class
- SetPreparedResponse.class

## 12 Ejercicio 12

### Enunciado:

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`

### Respuesta a la cuestión:

Simplemente utilizamos la funcionalidad `exec` en el fichero **build.xml** en el *target* `generar-stubs`:



```
<!-- web.xml -->

<target name="generar-stubs" depends="montar-jerarquia" description="Genera los stubs del cliente a
partir del archivo WSDL">
  <delete file="${build}/${tmpvisaclientjar}" />
  <jar jarfile="${build}/${tmpvisaclientjar}" >
    <fileset dir="${build.client}/WEB-INF/classes" />
  </jar>
  <move file="${build}/${tmpvisaclientjar}" todir="${build.client}/WEB-INF/lib" />
  <exec executable="/bin/sh">
    <arg value="-c"/>
    <arg value="wsimport -d ${build.client}/WEB-INF/classes -p ${paquete}.visa ${wsdl.url}"/>
  </exec>
</target>
```

### 13 Ejercicio 13

Enunciado:

- Realice un despliegue de la aplicación completo en dos nodos tal y como se explica en la Figura 8. Habrá que tener en cuenta que ahora en el fichero build.properties hay que especificar la dirección IP del servidor de aplicaciones donde se desplegará la parte del cliente de la aplicación y la dirección IP del servidor de aplicaciones donde se desplegará la parte del servidor. Las variables as.host.client y as.host.server deberán contener esta información.
- Probar a realizar pagos correctos a través de la página testbd.jsp. Ejecutar las consultas SQL necesarias para comprobar que se realiza el pago. Anotar en la memoria práctica los resultados en forma de consulta SQL y resultados sobre la tabla de pagos. Incluye evidencias en la memoria de la realización del ejercicio

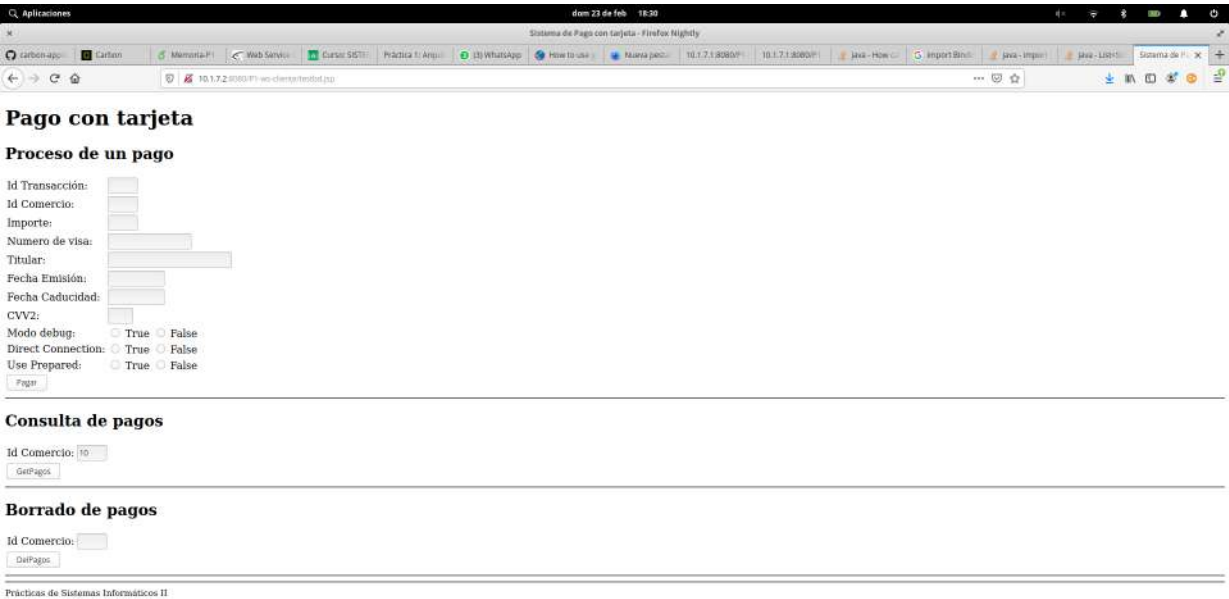
Respuesta a la cuestión:

Cambiamos las variables de entorno del fichero **build.properties**, de tal forma que **as.host.client** pasa a valer **10.1.7.2** y **as.host.server** sigue siendo **10.1.7.1**, igual que la base de datos. A continuación compilamos, empaquetamos y desplegamos el cliente (**ant cliente**). Ahora podemos acceder a la funcionalidad del servidor de aplicaciones desde la página **http://10.1.7.2:8080/P1-ws-cliente** e intentar realizar un pago:

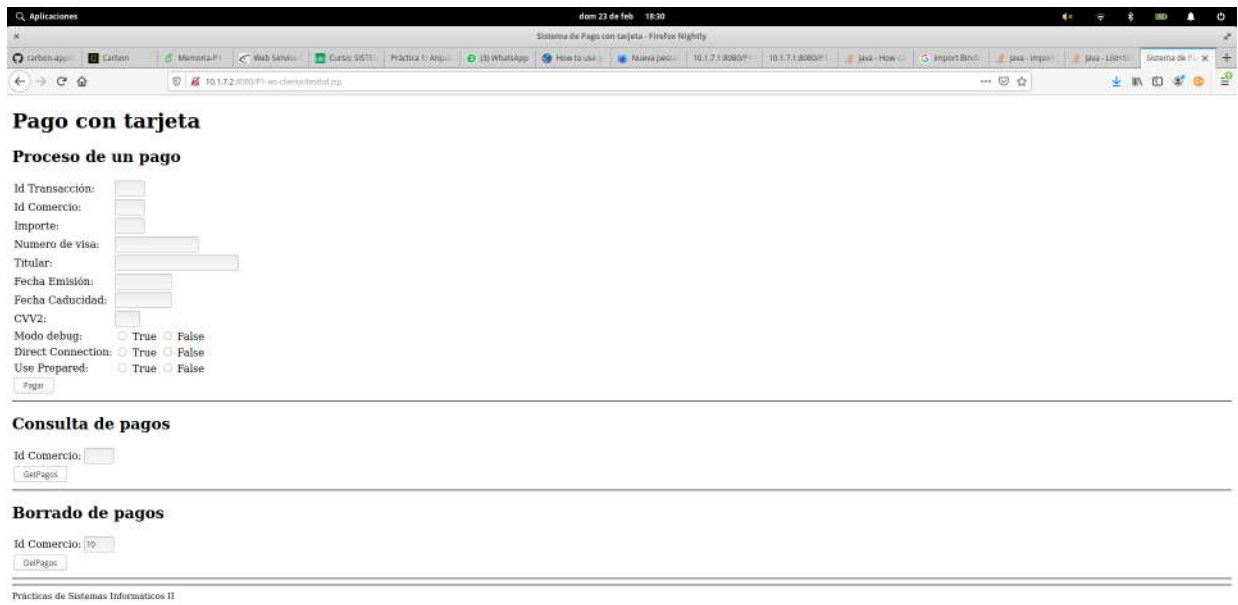




Como veamos a continuación, el pago se ha realizado exitosamente. Ahora vamos a hacer lo mismo, pero desde la página de *debug*: <http://10.1.7.2/P1-ws-cliente/testbd.jsp>. Empezamos probando la funcionalidad de consulta de pagos realizados:



También aprovechamos y probamos la opción de borrado de pagos:



Ahora volvemos a probar que nuestro servicio funciona correctamente, pero esta vez desde esta página:



Todo parece haber ido según lo esperado. Comprobemos esto último directamente desde la base de datos utilizando *ssh*:

```
dom 23 de feb 18:32
ssh si2@10.1.7.1
si2@si2srv01:~$ psql -U alumodb -d visa
psql (8.4.10)
Type "help" for help.

visa=# SELECT * FROM pago;
 idautorizacion | idtransaccion | codresguesta | importe | idcomercio | numerotarjeta | fecha
-----
3 | 1 | 800 | 100 | 10 | 1111 2222 3333 4444 | 2020-02-23 09:32:00,718827
(1 row)

visa=#
```



## 14 Cuestiones

### 14.1 Cuestión 1

#### Enunciado:

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.

#### Respuesta:

Se introduce el pago en pago.html, se envía y se pasa por el servlet `ComienzaPago.java`. Aquí se comprueba que los datos del formulario de pago.html, sean correctos y se sirve al cliente web el fichero jsp `formdatosvisa.jsp`. Al rellenar los datos y enviarlos, se lanza el servlet `ProcesaPago.java` donde se comprueba que la fecha ha expirado (antes de comprobar en la base de datos que la tarjeta sea o no correcta), entonces se redirige a `formdatosvisa.jsp` indicando que la fecha es incorrecta.

### 14.2 Cuestión 2

#### Enunciado:

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?

#### Respuesta:

El servlet `ComienzaPago.java` sirve `formdatosvisa.jsp`, que es necesario para solicitar la información sobre el pago con tarjeta. Al enviar este formulario se invoca al servlet `ProcesaPago.java` que es el encargado de procesarla con diversas llamadas a `VisaDAO`.

### 14.3 Cuestión 3

#### Enunciado:

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?

#### Respuesta:

pago.html contiene la información para solicitar los datos del pago (id, idComercio e importe). Estos son procesados por el servlet `ComienzaPago.java`. Esta información se almacena en la sesión (en una instancia de `PagoBean`), posteriormente, se sirve el jsp `formdatosvisa.jsp` que solicita la información de la tarjeta. Una vez se envía este formulario, el servlet `ProcesaPago.java` se encarga de procesar los datos de la tarjeta (almacenandolos en un `TarjetaBean`) y, si los datos son correctos, comparte estos datos y el objeto `PagoBean` con `VisaDAO`. `VisaDAO` tiene acceso a los datos del pago y de la tarjeta. Tras realizar la comprobación de que la tarjeta esté autorizada, este módulo se pone en contacto con la base de datos PSQL y almacena la información del pago.

### 14.4 Cuestión 4

#### Enunciado:

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?

#### Respuesta:

Cuando se utiliza la página de pruebas extendida `testbd.jsp`, accedemos directamente al servlet `ProcesaPago.java`, de esta forma evitamos invocar `ComienzaPago.java` y no se nos sirve `formdatostarjeta.jsp`.

Sigue funcionando el pago ya que la única diferencia es que `ComienzaPago.java` instancia un `PagoBean` con la información del pago. Como ahora no pasamos por `ComienzaPago`,

en `ProcesaPago` existe un control para comprobar que la variable de sesión con la información del pago sea distinta de `null`. Si no está inicializado (en caso de que lleguemos por `testbd.jsp`), lo inicializa con la información solicitada.

### III Conclusiones

El objetivo fundamental de esta práctica era adentrarse en la arquitectura de JAVA EE desde el punto de vista del arquitecto de software. Adicionalmente, hemos alcanzado los siguientes subobjetivos:

- Experimentar con un sistema multicapa (multitier) de varios niveles: interfaz cliente, aplicación servidora y base de datos. La aplicación servidora la subdividiremos en varios niveles según el proyecto adquiera más complejidad.
- Introducir la aplicación de ejemplo que emplearemos a lo largo de todas las prácticas: Aplicación VISA para venta electrónica. Esta aplicación hace uso de JSP, Servlets y JavaBeans.
- Conocer JDBC como API de acceso a base de datos.
- Conocer y experimentar con la tecnología de publicación de Servicios Web o Web Services
- Automatización de tareas de construcción y despliegue con la herramienta ant.

Ahora avanzamos a la segunda parte de esta práctica, donde podremos continuar profundizando en elementos de la arquitectura JAVA EE.

### IV Bibliografía

- Java EE 7 Tutorial
- API Java EE 7
- Documentación P1A Moodle