Capítulo 6

Servidor Georap

"Ahora ven, escríbelo en una tablilla, grábalo en un libro, y que dure hasta el último día para testimonio"

Sagrada Biblia, Isaías 30:8

6.1 Introducción

Hasta ahora hemos logrado nuestro objetivo de conseguir un software capaz de realizar correcciones RTCM sobre las coordenadas obtenidas por un navegador de código, las cuales previamente habían sido enviadas a nuestra PDA, pero llegado este punto observamos la posibilidad de que esas medidas puedan ser enviadas a través de Internet para que otra persona pueda acceder a nuestra posición en el momento que estime oportuno, o simplemente para dejar almacenada de forma fija nuestra posición en una base de datos porque se esté recopilando información geográfica. Con esta finalidad se ha programado una página web dinámica mediante JSP (*JavaServer Pages*), que está interconectada a una base de datos MySQL, que alojada en un servidor registra y almacena las posiciones enviadas por los usuarios de GEORAP CE. Cualquier persona, desde cualquier parte del mundo puede acceder a una pagina web que mostrará mediante un mapa de Google Maps la situación de las posiciones registradas en la base de datos así como todas las entradas en dicha base de datos, de este modo no solo podremos localizar rápidamente la situación exacta del usuario sino que podremos hacer un seguimiento de cómo se ha ido desplazando, además la página web detecta la hora a la que se envían los datos por lo que podremos saber con exactitud, quien y en que momento se encontraba en un determinado lugar. Por último, la base de datos está dotada de una contraseña de modo que solo el administrador del servidor podrá gestionar las posiciones como las entradas que ya considere obsoletas.

6.2 Descripción de tecnologías

6.2.1 JSP

Para programar la página web hemos utilizado el lenguaje JSP (JavaServer Pages) desarrollado por Sun Microsystems. Este formato nos da la posibilidad de insertar código dinámico que trabaja en conjunto con código HTML, haciendo así que el código HTML sea más funcional, por ejemplo incorporando consultas a bases de datos (como ha sido nuestra intención). La forma de proceder del servidor de aplicaciones será la de interpretar el código contenido en la página JSP para así construir el código Java del *servlet* a generar, este *servlet* generará entonces el documento HTML que se presenta en la pantalla del navegador.

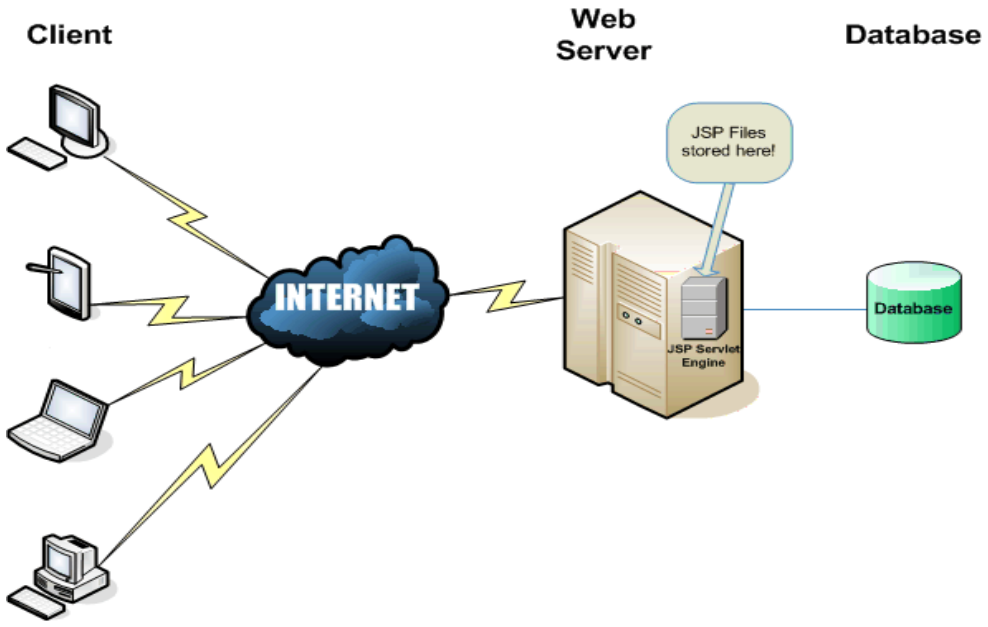

Figura 1. Funcionamiento de la tecnología JSP

6.2.2 MySQL

La base de datos utilizada para interactuar con nuestra página web dinámica y alojada en nuestro servidor es MySQL, una base de datos relacional, es decir, una representación lógica de datos que permite acceder a éstos sin necesidad de considerar la estructura física de los mismos.

MySQL es un sistema de administración de bases de datos. Una base de datos es una colección estructurada de tablas que contienen datos. Ésta puede ser desde una simple lista de compras a una galería de pinturas o el vasto volumen de información en una red corporativa. Para agregar, acceder, y procesar datos guardados en un computador, se necesita un administrador como MySQL Server. Dado que los computadores son muy buenos manejando grandes cantidades de información, los administradores de bases de datos juegan un papel central en computación, como aplicaciones independientes o como parte de otras aplicaciones. Se trata en definitiva de un sistema de administración relacional de bases de datos. Una base de datos relacional archiva datos en tablas separadas en vez de colocar todos los datos en un gran archivo. Esto permite velocidad y flexibilidad. Las tablas están conectadas por relaciones definidas que hacen posible combinar datos de diferentes tablas sobre pedido.

Por último cabe resaltar que es software de fuente abierta. Fuente abierta significa que es posible para cualquier persona usarlo y modificarlo. Cualquier persona puede bajar el código fuente de MySQL y usarlo sin pagar. Cualquier interesado puede estudiar el código fuente y ajustarlo a sus necesidades. MySQL usa licencia GPL (GNU General Public License) para definir que puede hacer y que no puede hacer con el software en diferentes situaciones.

6.3 Descripción global del Servidor

Nuestro servidor se compone de cinco páginas JSP:

- Index.jsp
- Posicion.jsp
- DatabaseVisor.jsp
- Posicionnodb.jsp
- ErrorProcessor.jsp

y de dos archivos *.class* que han sido generados a partir de la compilación de sus correspondientes archivos *.java*:

- GeoBean.class
- GeoDataBean.class

6.3.1 Página Index.jsp

Esta página despliega un formulario en nuestro navegador que nos solicita nuestro nombre y las coordenadas de longitud y latitud que hemos obtenido mediante Georap CE.

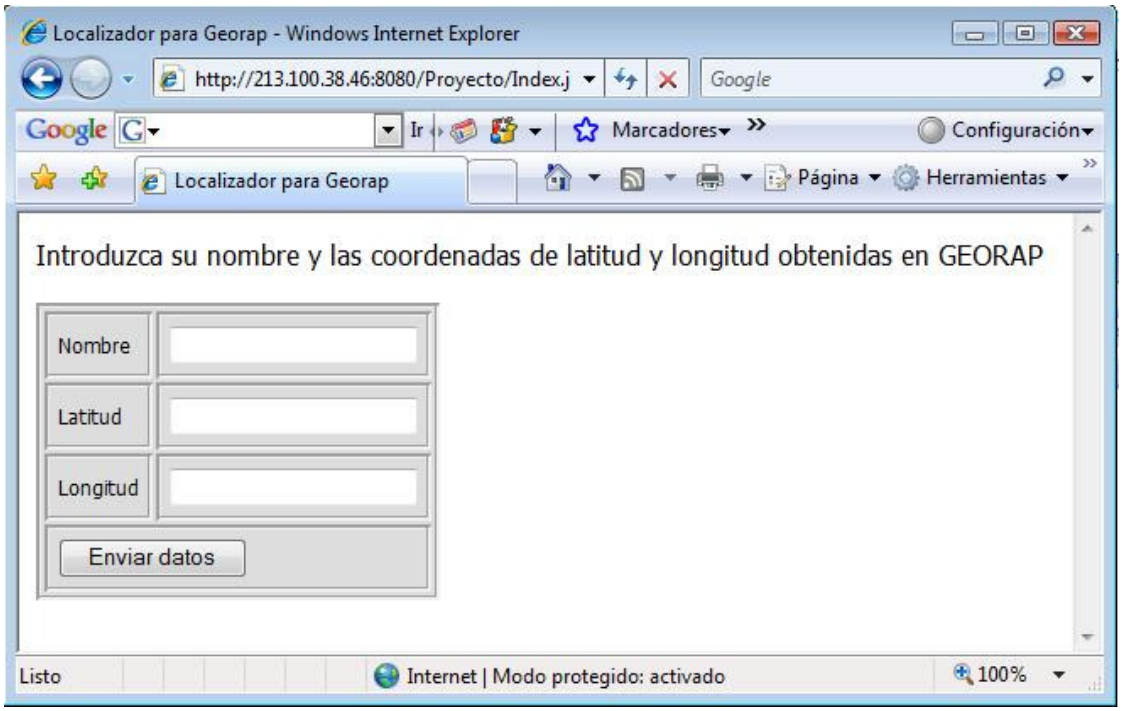

Figura 2. Pantalla de navegador mostrando Index.jsp

Una vez introducidos los datos, pulsamos el botón enviar, en ese instante se producen varias acciones transparentes al usuario; en primer lugar se captura la fecha y hora en la que se produjo la consulta, en segundo lugar los datos de nombre, latitud, longitud y fecha/hora son almacenados en las correspondientes variables declaradas en la clase *GeoBean.class*, seguidamente la clase *GeoDataBean.class* recoge dichos datos y los escribe en la correspondiente tabla de la base de datos MySQL accediendo a ella mediante un nombre de usuario y una contraseña. Aquí es importante resaltar que

como desarrollador de este proyecto he programado la clase *GeoDataBean.class* para que almacene los datos en la tabla "posicion" perteneciente a la base de datos "proyecto" donde los campos de posicion son "nombre, latitud, longitud, date", además de acceder a la base de datos mediante el username "root" y el password "jamon", es importante conservar dichos parámetros en el servidor de lo contrario no funcionará, en el caso de que el operador quisiera utilizar otro nombres o contraseña habría que modificar ciertas líneas de *GeoDataBean.java* y recompilarlo. Los campos a modificar serán explicados en detalle más adelante en el manual del operador del servidor.

Por último *Index.jsp* nos redireccionará automáticamente a *Posicion.jsp*, que describiremos a continuación.

6.3.2 Página Posicion.jsp

Esta página recibe los datos de nuestra posición y nos muestra donde nos encontramos mediante la API que ofrece Google Maps, esta API incluye todas las funciones de Google Maps, en ella podremos cambiar el modo de visualización, disponemos también de la barra de zum así como la de desplazamiento y la escala.

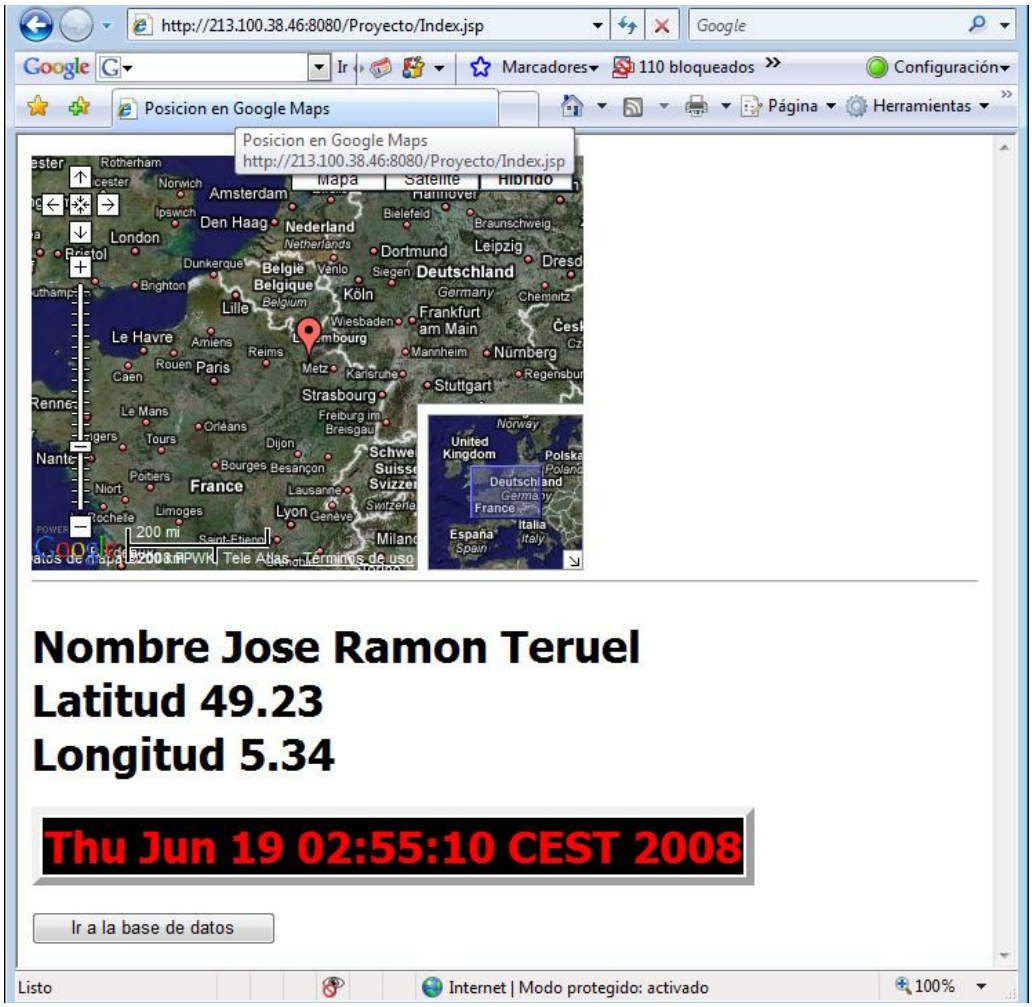

Figura 3. Navegador mostrando Posicion.jsp

Además nos muestra los datos que fueron enviados a la base de datos así como la fecha y hora en que se registro nuestra entrada. Por último el botón "Ir a la base de datos" nos da la opción de acceder a la página que muestra los registros de base de datos. Al pulsar este botón seremos direccionados a *DatabaseVisor.jsp*.

6.3.3 Página DatabaseVisor.jsp

A esta página podemos acceder desde *Posición.jsp* o directamente a través de su URL si no somos usuarios de Georap, o no lo estamos usando en este momento, sino que lo único que queremos es acceder a los datos registrados en la base de datos de localizaciones.

Esta página es la más compleja, mediante un código java y utilizando la clase *GeoDataBean.class* hace un recorrido por la base de datos discriminando los usuarios y asignando a cada uno un identificador. En la parte superior de la página aparece un mapa de Google Maps donde se encuentran ubicados todos los registros de que estén en ese momento almacenados en la base de datos, los puntos con el mismo número se corresponden con entradas donde el usuario es el mismo.

Figura 4. Mapa mostrando las posiciones registradas en la base de datos

Además esta ventana de Google Maps nos muestra los datos de la posición, así como el usuario y la hora pinchando encima del puntero que nos interese.

Figura 5. Datos mostrados al pinchar sobre el puntero

Bajo el mapa esta página muestra todas las entradas que hay en ese momento registradas en la base de datos, en forma de tabla, y ordenadas por criterio de fecha y hora.

Entradas en la base de datos

Nombre	Latitud	Longitud	Date
Jose Ramon Teruel	36.25	-2.31	4 de abril de 2008 3:17:15 CEST
Alejandro Villegas	37.22	-5.11	4 de abril de 2008 3:18:11 CEST
paco	55	56	4 de abril de 2008 3:23:21 CEST
paco	54	-89	4 de abril de 2008 3:23:47 CEST
Jose Ramon Teruel	39.12	-2.56	4 de abril de 2008 18:59:39 CEST

Figura 6. Tabla que muestra las entradas en la base de datos

Por último se ha introducido en esta página la posibilidad de localizar una ubicación especifica rellenando el formulario inferior que nos solicita las coordenadas de latitud y longitud que queremos visualizar, al pulsar el botón "Enviar consulta" seremos enviados a la página *Posicionnodb.jsp*.

6.3.4 Página Posicionnodb.jsp

Esta página es la más sencilla y se limita a mostrar por pantalla la posición solicitada ubicándola en un mapa, así como los datos introducidos mediante la consulta.

6.3.5 Página ErrorProcessor.jsp

Esta página se encarga de procesar los errores que puedan producirse, determinar a que causa se deben (Error al interactuar con la base de datos, error por no poder comunicarse con las clases, etc…) y muestra por el navegador una página en la cual aparece un mensaje indicando que tipo de error se produjo.

6.3.6 GeoBean.class

Esta clase se utiliza como punto de almacenamiento intermedio entre la web y la base de datos. En ella están declaradas cuatro variables que almacenaran los campos nombre, latitud, longitud y fecha, así como un método *set* y otro *get* por cada una de ellas, que permiten definir el valor de la variable (método *set*) u obtenerlo (método *get*). *Index.jsp* utiliza la invocación al método *set* para guardar los valores rellenado en el formulario y la clase GeoDataBean utiliza el método *get* para obtener el valor de esas variables y rellenar la base de datos. Además, GeoDataBean también utiliza el método *set* para enviar a través de GeoBean los datos ya almacenados en la base de datos a la página *DatabaseVisor.jsp*.

6.3.7 GeoDataBean.class

Esta clase es más compleja que la anterior y es la responsable de interactuar con la base de datos, podemos resumir que realiza tres funciones fundamentales:

1. Carga el driver *jdbc* que nos permite interactuar con MySQL, define que base de datos va a utilizar y almacena el nombre de usuario y la contraseña, necesarios para poder acceder a los datos.

2. Cuando consultamos la página *DatabaseVisor.jsp* realiza la lectura completa de la tabla definida y los va mandando a la página web uno por uno utilizando como almacenamiento intermedio las variables declaradas en la clase *GeoBean.class*.

3. Cuando rellenamos el formulario en *Index.jsp* se encarga de crear una nueva entrada en la tabla con los datos enviados.

6.4 Manual del Servidor

En esta parte vamos a explicar detalladamente y paso a paso como poner nuestro servidor Georap a funcionar en nuestro equipo.

6.4.1 Software necesario

Debido a que programamos las páginas en JSP necesitaremos que nuestro servidor sea contenedor de *servlets,* necesitaremos instalar Apache Tomcat independientemente de si queremos o no que sea nuestro servidor principal, como base de datos utilizaremos MySQL y además tendremos que usar un conector para poder hacerlos funcionar juntos. A continuación explicaremos el procedimiento correcto de instalación de estos componentes.

1. El primer paso será instalar Apache Tomcat en nuestro equipo. Tomcat es gratuito y podemos descargarlo desde su página oficial: http://tomcat.apache.org/ (julio 2008). La versión que hemos usado en el desarrollo de este proyecto es la 6.0.16, que es la que funciona con la distribución 2.1 de JSP.

2. Una vez descargado el instalador lo ejecutamos y procedemos a la instalación del producto. El instalador nos irá guiando a través del proceso y nos irá solicitando la siguiente información: tipo de instalación, ruta del

programa, nombre y contraseña para el administrador del servidor, puerto por el que queremos que se procesen las peticiones y la ruta donde tenemos instalada la maquina virtual de java (aquí mencionaré que como resulta obvio previo a todo este proceso debemos tener instalada la maquina virtual de java, en el desarrollo de este proyecto se ha usado la versión jre1.6.0_06). Por último solicitaremos en la última ventana antes de cerrar la instalación que se lance el servicio. Para comprobar que todo el proceso ha sido correcto abriremos un navegador con la dirección http://localhost:8080/ (en caso de haber elegido otro número de puerto durante la instalación, sustituirlo por el 8080 en la URL anterior), si todo el proceso fue bien debería mostrarnos lo siguiente por pantalla.

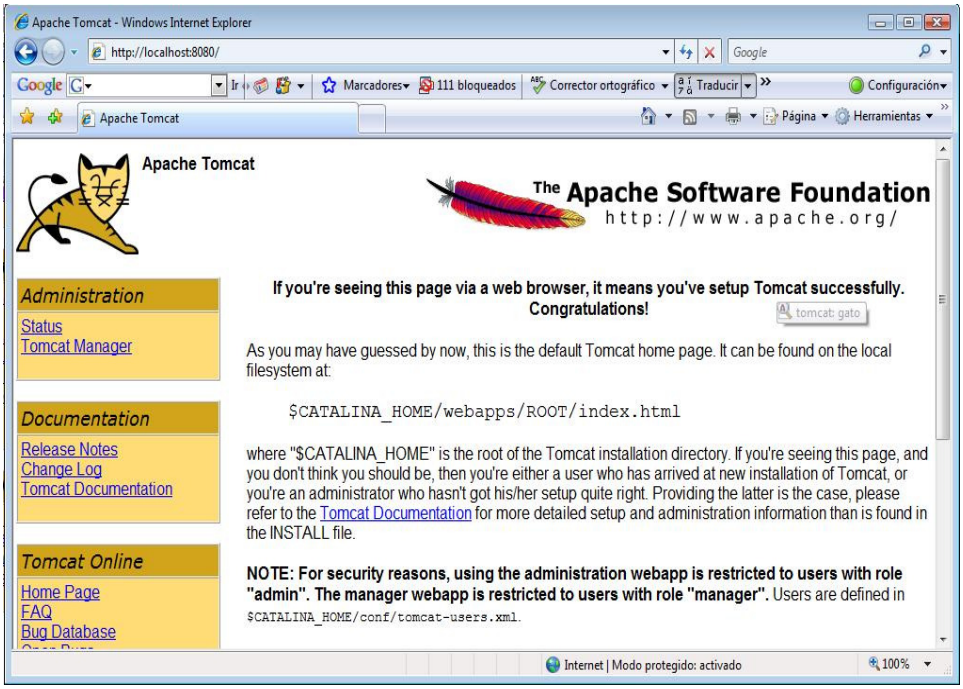

Figura 7. Pantalla principal de Tomcat

3. A continuación necesitaremos instalar la base de datos MySQL, podemos hacerlo de forma gratuita en http://dev.mysql.com/downloads/ (julio 2008). La versión que hemos utilizado ha sido la 6.0 que es la única que funciona con Microsoft Windows Vista. A continuación la instalaremos siguiendo los pasos del instalador.

4. Como última tarea software necesitamos conectar MySQL con Tomcat para esto será necesario ubicar el conector *mysql-connector-java-5.1.6-bin.jar* desde http://dev.mysql.com/downloads/connector/j/5.1.html (julio 2008). Este fichero debemos ubicarlo dentro de la carpeta \lib ubicada en la instalación de Apache Tomcat.

5. Por último, debemos modificar las variables de entorno de nuestro sistema operativo. En el Path del sistema debemos añadir: C:\ProgramFiles\MySQL\MySQL Server 6.0\bin (siendo esta la ruta donde se encuentra la carpeta bin de MySQL) y en la variable CLASSPATH debemos ubicar el conector de esta manera c:\tomcat6\lib\mysql-connector-java-5.1.6-bin (o modificando la ruta en caso de que instalásemos Tomcat en una carpeta diferente).

6.4.2 Ubicación de los archivos

En la carpeta *Servidor* de este proyecto se encuentran todos los archivos que se han programado y todo lo necesario para su correcto funcionamiento. Estos archivos deben ubicarse dentro de una carpeta en el directorio webapps de Tomcat. La estructura del árbol de directorios que se distribuye es la siguiente:

- \Servidor\
    - Index.jsp
    - Posicion.jsp
    - DatabaseVisor.jsp
    - Posicionnodb.jsp
    - ErrorProcessor.jsp
    - \WEB-INF\
        - Web.xml
        - \lib\
            - mysql-connector-java-5.1.6-bin
        - \classes\
            - \codigo\
                - \beans\
                    - GeoBean.class
                    - GeoDataBean.class

Es muy importante que esta estructura se mantenga intacta dentro de la carpeta *webapps*\ de Tomcat para que el programa funcione correctamente, por lo que se

recomienda copiar y pegar directamente la carpeta Servidor en el directorio mencionado.

Como último paso para que el programa funcione debemos entrar en el monitor de MySQL y crear una base de datos llamada proyecto y dentro de ella una tabla llamada posición.

Llegados a este punto todo está preparado para dar servicio a los usuarios de Georap CE, para comprobar que todo el proceso anterior se ha llevado a cabo con éxito abriremos una ventana del navegador e introduciremos la dirección http://localhost:8080/Proyecto/DatabaseVisor.jsp, y por pantalla deberá aparecernos la página web que nos da información acerca de la posición de todos los usuarios en Google Maps y las entradas en la base de datos. Cualquier persona ajena al servidor podrá acceder a esta información a través de la dirección que asigne para esta ruta el administrador de la red donde está ubicado el software.

6.5 Manual Avanzado

6.5.1 Resolución de posibles conflictos

Durante la fase de pruebas del servidor Georap se ha detectado un conflicto que resultaba estar originado por la incompatibilidad de algunas librerías de la máquina virtual de Java con librerías de Apache Tomcat, debido a este conflicto no se podía cargar la página web. Al intentar cargar la página JSP en el navegador obteníamos un mensaje de error tal como éste:

type Exception report

message

description The server encountered an internal error () that prevented it from fulfilling this request.

exception

org.apache.jasper.JasperException: Unable to compile class for JSP:

An error occurred at line: 22 in the generated java file
The method getJspApplicationContext(ServletContext) is undefined for the type JspFactory

An error occurred at line: 82 in the generated java file
The method handlePageException(Exception) in the type PageContext is not applicable for the arguments (Throwable)

Stacktrace:
org.apache.jasper.compiler.DefaultErrorHandler.javacError(DefaultErrorHandler.java:92)
org.apache.jasper.compiler.ErrorDispatcher.javacError(ErrorDispatcher.java:330)
org.apache.jasper.compiler.JDTCompiler.generateClass(JDTCompiler.java:423)
org.apache.jasper.compiler.Compiler.compile(Compiler.java:308)
org.apache.jasper.compiler.Compiler.compile(Compiler.java:286)
org.apache.jasper.compiler.Compiler.compile(Compiler.java:273)
org.apache.jasper.JspCompilationContext.compile(JspCompilationContext.java:566)
org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:317)
org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:320)
org.apache.jasper.servlet.JspServlet.service(JspServlet.java:266)
javax.servlet.http.HttpServlet.service(HttpServlet.java:865)

…

Este error era se debía a que en Tomcat 6 las clases para la API JSP están contenidas en jsp-api.jar y las clases para la API servlet están contenidas en servlet-api.jar, ambas ubicadas en la carpeta lib perteneciente a la instalación de Tomcat. Estas clases están preparadas para resolver las distribuciones JSP 2.1 y Servlet 2.5, pero sin embargo nuestra distribución de java contenía el archivo servlet.jar que no es apto para esta versión ya que solo puede resolver las distribuciones JSP 1.1 y Servlet 2.2.

Por tanto, la forma de resolver este error es localizar la librería servlet.jar y eliminarla de cualquier carpeta de nuestra maquina virtual Java o instalación de Apache Tomcat.

6.5.2 Key de Google Maps

Como ya hemos visto hasta este momento, nuestro servidor utiliza en varias de sus pantallas un mapa de Google Maps, esta API se distribuye directamente por Google con un código que puede incluirse libremente en cualquier página HTML para generar el mapa. Pero este código incluye una clave que solo lo hace disponible para una determinada dirección de red, esto quiere decir que si portamos los archivos a un servidor distinto al que corresponde a la clave solicitada el mapa no cargará y obtendremos un mensaje de error como el que muestra la figura 8.

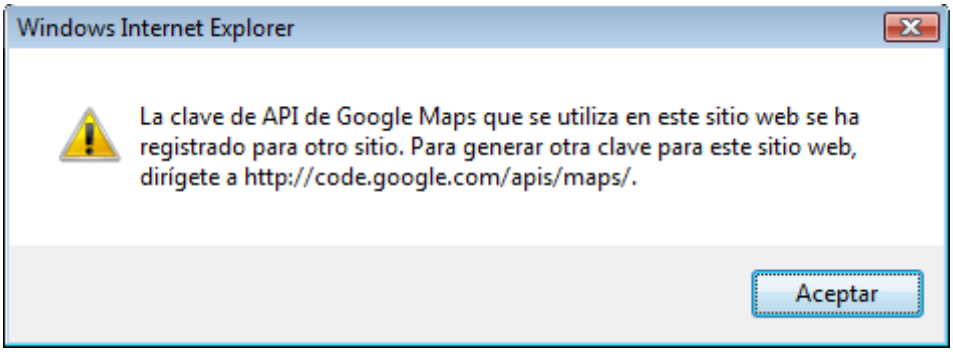

Figura 8. Error generado al usar una key de Google Maps incorrecta

Tal y como podemos ver en el diálogo deberemos generar una nueva clave de red a través del sitio web http://code.google.com/apis/maps/. En esta página debemos dirigirnos a la opción "*Sign up for a Google Maps API key"* y una vez ahí introducir la dirección URL para la que queramos que sea compatible nuestra clave. Tras rellenar este cuadro de diálogo y aceptar los términos y condiciones de uso establecidos por Google nos aparecerá una pantalla con nuestra clave, la URL para la cual nuestra clave es válida y un ejemplo de código HTML que podemos insertar en nuestra web para obtener la versión más sencilla de la API.

Por tanto si queremos utilizar nuestro servicio en otro servidor debemos cambiar las correspondientes keys en *Posicion.jsp*, *DatabaseVisor.jsp* y *Posicionnodb.jsp*, que se encuentran en la siguiente línea de código resaltada:

```
<meta http-equiv="content-type" content="text/html; charset=utf-8"/>
    <title>Google Maps JavaScript API Example</title>
<script
src="http://maps.google.com/maps?file=api&amp;v=2&amp;key=ABQIAAAAd2
zTJ0Rl4hTInXeaQ4WCRhRCfloyoF1ltM9lvoEy4sW1v9G0RRSyIeHAn25PB3ov2
1EwtWfrJ4hxg"

      type="text/javascript"></script>
    <script type="text/javascript">
```

Código 1. Extracto de código donde se ubica la clave de Google Maps

6.5.3 Manual básico SQL

Nuestro servicio ha sido programado para funcionar con una base de datos MySQL pero antes de empezar a funcionar el administrador del servidor debe haber creado una base de datos con unas características concretas para que el servicio pueda funcionar.

Nuestras páginas JSP requieren una serie de condiciones en la base de datos que son las siguientes:

- El nombre de usuario debe ser “root” y la contraseña “jamon”.
- La base de datos debe llamarse “proyecto”.
- Dentro de la base de datos “proyecto” debe encontrarse una tabla llamada “posicion”.
- La tabla posición debe tener cuatro campos: “nombre”, “latitud”, “longitud” y “date”.

Además de poder crear este entorno el administrador del servidor debe conocer la forma de operar con esta base de datos para poder gestionar todos los eventos. Por ejemplo debe saber como eliminar los registros que quedan ya obsoletos o que han sido enviados por usuarios que no nos interesa almacenar. Por este motivo se ha incluido un pequeño resumen de las principales sentencias a conocer con su correspondiente sintaxis.

CREATE DATABASE

Sintaxis:

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
[create_specification [, create_specification] ...]

create_specification:
    [DEFAULT] CHARACTER SET charset_name
  | [DEFAULT] COLLATE collation_name
```

CREATE DATABASE crea una base de datos con el nombre dado. Para usar CREATE DATABASE se necesita el privilegio CREATE en la base de datos.

En MySQL las bases de datos se implementan como directorios que contienen los ficheros correspondientes a las tablas de la base de datos. Como no hay tablas en una base de datos cuando esta es creada, la sentencia CREATE DATABASE sólo crea un directorio bajo el directorio "data" de MySQL.

CREATE TABLE

Sintaxis:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
  [(definición_create,...)]
  [opciones_tabla] [sentencia_select]
```

Sintaxis definición create:

```
   definición_columnas
  | [CONSTRAINT [símbolo]] PRIMARY KEY (index_nombre_col,...)
  | KEY [nombre_index] (nombre_col_index,...)
  | INDEX [nombre_index] (nombre_col_index,...)
  | [CONSTRAINT [símbolo]] UNIQUE [INDEX]
       [nombre_index] [tipo_index] (nombre_col_index,...)
```

```
  | [FULLTEXT|SPATIAL] [INDEX] [nombre_index] (nombre_col_index,...)
  | [CONSTRAINT [símbolo]] FOREIGN KEY
       [nombre_index] (nombre_col_index,...) [definición_referencia]
 | CHECK (expr)
```

Sintaxis de definición columnas:

```
nombre_col tipo [NOT NULL | NULL] [DEFAULT valor_por_defecto]
        [AUTO_INCREMENT] [[PRIMARY] KEY] [COMMENT 'string']
        [definición_referencia]
```

Sintaxis de tipo:

```
TINYINT[(longitud)] [UNSIGNED] [ZEROFILL]
  | SMALLINT[(longitud)] [UNSIGNED] [ZEROFILL]
  | MEDIUMINT[(longitud)] [UNSIGNED] [ZEROFILL]
  | INT[(longitud)] [UNSIGNED] [ZEROFILL]
  | INTEGER[(longitud)] [UNSIGNED] [ZEROFILL]
  | BIGINT[(longitud)] [UNSIGNED] [ZEROFILL]
  | REAL[(longitud,decimales)] [UNSIGNED] [ZEROFILL]
  | DOUBLE[(longitud,decimales)] [UNSIGNED] [ZEROFILL]
  | FLOAT[(longitud,decimales)] [UNSIGNED] [ZEROFILL]
  | DECIMAL(longitud,decimales) [UNSIGNED] [ZEROFILL]
  | NUMERIC(longitud,decimales) [UNSIGNED] [ZEROFILL]
  | DATE
  | TIME
  | TIMESTAMP
  | DATETIME
  | CHAR(longitud) [BINARY | ASCII | UNICODE]
  | VARCHAR(longitud) [BINARY]
  | TINYBLOB
  | BLOB
  | MEDIUMBLOB
  | LONGBLOB
  | TINYTEXT
  | TEXT
  | MEDIUMTEXT
  | LONGTEXT
  | ENUM(valor1,valor2,valor3,...)
  | SET(valor1,valor2,valor3,...)
  | tipo_spatial
```

Sintaxis de nombre col index:

```
nombre_col [(longitud)] [ASC | DESC]
```

Sintaxis de definición de referencia:

```
REFERENCES nombre_tbl [(nombre_col_index,...)]
                    [MATCH FULL | MATCH PARTIAL | MATCH SIMPLE]
                    [ON DELETE opción_referencia]
                    [ON UPDATE opción_referencia]
```

Sintaxis de opción referencia:

```
RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT
```

Sintaxis de opciones tabla:

```
opción_tabla [opción_tabla] ...
```

Sintaxis de opción_tabla:

```
{ENGINE|TYPE} = {BDB|HEAP|ISAM|InnoDB|MERGE|MRG_MYISAM|MYISAM }
  | AUTO_INCREMENT = valor
  | AVG_ROW_LENGTH = valor
  | CHECKSUM = {0 | 1}
  | COMMENT = 'cadena'
  | MAX_ROWS = valor
  | MIN_ROWS = valor
  | PACK_KEYS = {0 | 1 | DEFAULT}
  | PASSWORD = 'cadena'
  | DELAY_KEY_WRITE = {0 | 1}
  | ROW_FORMAT = {
DEFAULT|DYNAMIC|FIXED|COMPRESSED|REDUNTANT|COMPACT}
  | RAID_TYPE = { 1 | STRIPED | RAID0 }
       RAID_CHUNKS=valor
```

```
      RAID_CHUNKSIZE=valor
  | UNION = (nombre_tabla,[nombre_tabla...])
  | INSERT_METHOD = { NO | FIRST | LAST }
  | DATA DIRECTORY = 'camino de directorio absoluto'
  | INDEX DIRECTORY = 'camino de directorio absoluto'
      | [DEFAULT] CHARACTER SET nombre_conjunto_caracteres [COLLATE
nombre_cotejo]
```

Sintaxis de sentencia select:

```
[IGNORE | REPLACE] [AS] SELECT ...    (Alguna sentencia select legal)
```

CREATE TABLE crea una tabla con el nombre dado. Se debe poseer el privilegio *CREATE* para la tabla.

Por defecto, la tabla se crea en la base de datos actual. Se producirá un error si la tabla ya existe, si no hay una base de datos actual o si la base de datos no existe.

DELETE

Sintaxis:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM table_name
       [WHERE where_definition]
       [ORDER BY ...]
       [LIMIT row_count]
```

DELETE elimina las columnas de “table_name” que satisfagan la condición dada por la “where_definition”, y devuelve el número de registros borrados.

Si se usa una sentencia DELETE sin la cláusula WHERE, todas las filas serán borradas.

INSERT

Sintaxis:

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
      [INTO] tbl_name [(col_name,...)]
      VALUES ({expression | DEFAULT},...),(...),...
      [ ON DUPLICATE KEY UPDATE col_name=expression, ... ]
```

Ó

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
      [INTO] tbl_name
      SET col_name={expression | DEFAULT}, ...
      [ ON DUPLICATE KEY UPDATE col_name=expression, ... ]
```

Ó

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
      [INTO] tbl_name [(col_name,...)]
      SELECT ...
```

INSERT inserta nuevas filas en una tabla existente. Los formatos INSERT … VALUES e INSERT … SET, insertan filas basándose en los valores especificados explícitamente. El formato INSERT … SELECT inserta filas seleccionadas de otra tabla o tablas.

Tbl_name es la tabla donde se insertarán las filas. Las columnas para las que la sentencia proporciona valores se pueden especificar de las siguientes formas:

- Las listas de nombres de columnas o la cláusula SET indican las columnas explícitamente.

- Si no se especifica una lista de columnas para INSERT … VALUES o INSERT … SELECT, se deben proporcionar valores para todas las columnas en la lista VALUES() o por SELECT. Si no se conoce el orden de las columnas dentro de la tabla, usar "DESCRIBE *tbl_name"* para encontrarlo.

SELECT

Sintaxis:

```
SELECT
      [ALL | DISTINCT | DISTINCTROW]
      [HIGH_PRIORITY]
      [STRAIGHT_JOIN]
      [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
      [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
   select_expr,...
   [INTO OUTFILE 'file_name' export_options]
    | INTO DUMPFILE 'file_name']
   [FROM table_references
      [WHERE where_definition]
      [GROUP BY {col_name | expr | position}
        [ASC | DESC], ... [WITH ROLLUP]]
      [HAVING where_definition]
      [ORDER BY {col_name | expr | position}
        [ASC | DESC] ,...]
      [LIMIT {[offset,] row_count | row_count OFFSET offset}]
      [PROCEDURE procedure_name(argument_list)]
      [FOR UPDATE | LOCK IN SHARE MODE]]
```

SELECT se usa para recuperar filas seleccionadas de una o más tablas.

- Cada *select_expr* indica una columna que se quiere recuperar.

- *Table_references* indica la tabla o tablas de las que se recuperan filas.

- *Where_definition* consiste en la palabra clave WHERE seguida por una expresión que indica la condición o condiciones que las filas deben satisfacer para ser seleccionadas.

SELECT puede usarse también para recuperar filas calculadas sin referencia a ninguna tabla.

USE

Sintaxis:

```
USE db_name
```

La sentencia USE *db_name* indica a MySQL que use la base de datos *db_name* como la base de datos por defecto en sentencias subsiguientes. La base de datos sigue siendo la base de datos por defecto hasta el final de la sesión o hasta que se use otra sentencia USE.

Éstas son las instrucciones básicas de MySQL con las cuales ya podremos realizar una gestión básica del servidor, y prepararlo para funcionar con nuestro servicio. No obstante las posibilidades de esta base de datos van mucho más lejos y existen multitud de instrucciones y procedimientos con muchas aplicaciones.

6.5.4 Adaptación del código

Llegado a este punto ya sabemos todo lo necesario para que nuestro servicio funcione en cualquier servidor, pero puede suceder que el servidor ya tuviera instalada otra versión de MySQL en la cual el usuario o la contraseña fueran diferentes a los que usa nuestro servicio, o quizás, por alguna razón no podemos usar los nombres "proyecto" o "posición" para la base de datos y la tabla. A continuación vamos a ver como deberíamos modificar el código para poder cambiar estos datos.

Como dijimos anteriormente nuestro servicio accederá a MySQL a través de la JavaBean *GeoDataBean.class*, donde se realiza la autentificación de usuario mediante el username y el password, por tanto si necesitáramos usar otros campos distintos a los por defecto debemos modificarlos en el archivo *GeoDataBean.java* que esta incluido en la distribución del servicio:

```
rowSet.setUsername( "root" );
rowSet.setPassword( "jamon" );
```

Código 2. Extracto de código de GeoDataBean.java donde se definen el username y el pass de la base de datos.

Por otra parte nuestro programa al grabar y leer en/de la base de datos siempre actuará en la tabla "posición" de la base de datos "proyecto", además de que los nombres de los campos esperados serán "nombre", "latitud", "longitud" y "date", esto también se realiza desde la clase *GeoDataBean.java*, por lo que si queremos cambiarlo debemos actuar modificando las partes resaltadas del siguiente código por los nombres que queramos para nuestra base de datos y nuestra tabla:

```
rowSet = new CachedRowSetImpl();
      rowSet.setUrl(
"jdbc:mysql://localhost/proyecto?autoReconnect=true&relaxAutoCommit=
true" );
      rowSet.setUsername( "root" );
      rowSet.setPassword( "jamon" );

        // obtain list of titles
      rowSet.setCommand(
         "SELECT nombre, latitud, longitud, date FROM
posicion" );
      rowSet.execute();
   } // end GeoDataBean constructor
```

Código 3. Fragmento de código donde se definen los nombres de la base de datos, la tabla y los campos de la tabla con los que se interactúa.