Desarrollo de aplicaciones orientadas a Internet

Desarrollo e implementación de una aplicación web para la gestión de reservas hosteleras

Web de Reservas

Luis Fernando Nicolás Alonso, Álvaro Fernández Fernández
01/01/2011

Índice

1. Introducción

El gran crecimiento en el uso de Internet entre la población en la sociedad actual ha provocado una revolución en la forma de potenciar y gestionar diferentes tipos de servicios, incluso los más tradicionales. Siguiendo esta línea innovadora, muchos negocios pretenden servirse de las ventajas que proporcionan las aplicaciones web para mejorar el uso y acceso a sus servicios. Este es el caso de un consorcio de restaurantes de Valladolid, cuya intención es conseguir un entorno intuitivo, sencillo y amigable desde el que un usuario pueda acceder a la consulta sobre la disponibilidad de un servicio, así como la reserva del mismo. Para ello, este consorcio espera conseguir la creación de un portal en el que recoger la disponibilidad para dar banquetes y otros servicios hosteleros en un determinado día.

En consecuencia, este documento pretende recoger los aspectos más importantes del desarrollo de una aplicación web para la gestión de los servicios de un consorcio de restaurantes de Valladolid. La aplicación en concreto se desarrolla bajo el nombre de *WebRes*, proporcionando a los clientes una serie de funcionalidades básicas como son la consulta de disponibilidad de servicio; la reserva de un servicio y la consulta del calendario de reservas de cada cliente.

La sección 2 de este informe presenta las clases Java principales del sistema desarrollado. En la sección 3 se recoge la información asociada a la base de datos que maneja el sistema. La sección 4 muestra algunos aspectos de diseño que han resultado conflictivos, así como la explicación de cómo se han solucionado y porqué. Finalmente, en la sección 5 se exponen las referencias empleadas en este documento y en el desarrollo de la aplicación web.

2. Clases principales del sistema

Las clases Java empleadas en la aplicación se encuentran agrupadas en 3 paquetes: *bean*, *datos* y *servlet*. A continuación se expone cada uno de los paquetes con sus principales clases, para finalizar esta sección realizando una pequeña descripción de las diversas JSPs utilizadas en la aplicación.

Paquete *bean*

El paquete *bean* recoge las clases que son necesarias para gestionar la aplicación. Básicamente acceden a la base de datos y extraen los datos necesarios para la ejecución de la aplicación en cada momento que serán almacenadas en las clases del paquete datos.

Quitando *ConexiónBDException.java, LoginException.java* y *ReservaExcepcion.java* que simplemente sirven para identificar el tipo de error que haya podido ocurrir en la aplicación, las principales clases de este paquete son las siguientes.

GestorReservas.java

Esta clase es utilizada por dos servlets, *WebRes.java* y *Ocupacion.java*.

Los dos primeros métodos de *GestorReservas.java,* que son *setDataGestor* y *setDataPublico* son usados por el servlet WebRes. Estos métodos extraen las reservas de

un mes (el seleccionado por el usuario o el actual por defecto) correspondientes a los gestores o a un usuario público respectivamente. Con la información de cada reserva se crea un objeto de la clase *Reserva,* que es almacenado en un vector. Ese vector es almacenado en una *Hashtable* llamada *reservas* cuya clave es la fecha de la reserva. De esta manera tenemos una tabla con 2 columnas, una es la fecha y la otra contiene un vector con todas las reservas de esa fecha. Así, al mostrar los calendarios se puede acceder a las reservas que tiene un usuario en cada fecha buscando en la *Hashtable* esa fecha. Si ese día no tiene ninguna reserva devolverá *null*.

Los métodos *setDataServicio* y *setDataSubServicio* realizan el mismo cometido que los métodos anteriores, pero en este caso extraen las reservas asociadas a un servicio o a un subservicio respectivamente. Además en este caso se extraen las reservas de dos meses, pues en el calendario de ocupación se muestran dos meses consecutivos. Estos métodos son utilizados por el servlet *Ocupacion*, que es el que gestiona las JSPs que muestran los calendarios de ocupación de los servicios y subservicios.

El método *reservar* se utiliza para modificar la tabla *DISPONIBILIDAD_SERVICIOS* de la base de datos. Primero se comprueba que los datos para realizar la reserva son correctos. Después se comprueba que no haya sido reservado antes por otro usuario, para evitar reservas simultaneas y finalmente se cambia el estado de la reserva a Ocupado y se asocia el login del usuario que haya realizado la reserva.

Finalmente el método *getReserva* devuelve el vector con las reservas de un día almacenadas en la *Hashtable*.

DispServBean.java

Esta clase fija la fecha de inicio y de fin de búsqueda de servicios, así como el identificador del tipo de servicio buscado y la descripción del mismo. Se utiliza por la JSP *sidebar* para la búsqueda de servicios.

InfoPerfil.java

Al igual que *GestorReservas.java*, esta clase extrae de la base de datos la información que se necesitará mostrar en el perfil del usuario, y permite a la JSP correspondiente (*perfil_gestor* o *perfil_publico*) acceder a esta información.

Si el usuario es público, se extraerá su identificador, su nombre, el número de reservas realizadas y su siguiente reserva. Además se comprueba si este usuario tiene dos reservas simultáneas para poder avisarle.

Si el usuario es gestor, se extrae su identificador de usuario, los servicios de los que es gestor, el número de reservas que están bajo su gestión, y cuál es la próxima reserva de la que tiene que ocuparse, informando si tiene reservas simultaneas.

Reserva.java

La clase *Reserva.java* almacena la información asociada a cada reserva: su identificador de disponibilidad, de servicio, de subservicio; el día, la hora y el estado de la misma; así como el login del usuario que la haya reservado. Esta clase permite a las JSPs acceder a la información de una reserva para mostrarla en los calendarios o en el perfil del usuario.

Paquete *datos*

Las clases de este paquete almacenan los datos necesarios por las JSPs para mostrárselos al usuario, permitiendo a estas acceder a ellos. Generalmente estas clases son instanciadas por las clases del paquete *bean*.

De nuevo, las clases *FechaExcepcion.java* e *IdTipoServicioExcepcion.java* permiten identificar los errores posibles.

DispServ.java

Esta clase es la utilizada para realizar la búsqueda de servicios y subservicios a través de la barra lateral de la aplicación.

El método *DispServ* extrae de la base de datos los servicios y subservicios existentes y disponibles entre las fechas de inicio y fin de búsqueda. Con esta información instancia las clases *Servicio.java, Subservicio.java* y *TiposServicio.java*, que serán accedidas por las JSPs para mostrar los datos correspondientes al usuario. Rellena un vector de servicios llamado *dispServ* que es la lista de los servicios disponibles entre las fechas deseadas.

El método *formateoFechas* sirve para que todas las fechas sigan el mismo formato que el utilizado en la base de datos (aaaa/mm/dd), aunque el usuario pueda introducirlo de otra forma.

El método *getNext* devuelve si hay más servicios en la lista de servicios disponibles entre las fechas de inicio y fin de búsqueda.

El método *getServicio* devuelve el siguiente servicio de la lista (es decir, del vector *dispServ*) y el método *isEmpty* devuelve si no hay ningún servicio disponible entre las fechas deseadas.

Servicio.java, Subservicio.java y TiposServicio.java

Como ya se ha comentado, estas clases almacenan los datos de los servicios, subservicios y tipos de servicios extraídos de la base de datos por *DispServ.java*. Contienen la información relativa al nombre, descripción, capacidad, identificadores, etc.... de los diversos servicios y subservicios de cada tipo de servicio.

Paquete *servlet*

En este paquete se almacenan los servlets que gestionan las JSPs servidas, la información de las request, y los beans con la información que precisa cada JSP.

WebRes.java

Este servlet es el encargado de servir las JSPs de inicio según el usuario sea gestor o público. Primero determina el rol del usuario a partir de la request, instancia un objeto de tipo *GestorReservas* para poder mostrar las reservas de ese usuario en un mes concreto, invoca el método correspondiente según el usuario sea gestor o público y sirve la JSP adecuada (*inicio* si es público, *inicioG* si es gestor).

DispServ.java

Este servlet gestiona la búsqueda de disponibilidad de servicios, devolviendo la JSP *dispServ* (que muestra la lista de servicios y subservicios disponibles) si el usuario es público.

Ocupacion.java

Se encarga, como *WebRes*, de instanciar un objeto *GestorReservas* para extraer las reservas asociadas a un servicio o subservicio, que serán mostradas en forma de calendario para el mes actual y el siguiente gracias a la JSP *ocupación*.

Perfil.java

Siguiendo la línea de *WebRes,* este servlet sirve las JSPs de los perfiles de los usuarios. Establece si el usuario es gestor o publico, instancia un objeto de tipo *InfoPerfil* para conseguir la información que se debe mostrar en el perfil del usuario. Si el usuario es gestor sirve la JSP *perfil_gestor*, y si el usuario es público sirve la JSP *perfil_publico*.

Reserva.java

Simplemente sirve la JSP que informa al cliente de que una reserva se ha realizado con éxito (*reserva*).

ConexionBD.java

Esta clase básicamente realiza las conexiones y desconexiones con la base de datos, a través de los métodos *CrearConexion* y *CerrarConexion*.

JSPs y hojas de estilo

A continuación se realiza una pequeña descripción de las JSPs y hojas de estilo usadas. La mayoría de las JSPs muestran la información requerida en la especificación de requisitos [1].

cabecera.jsp

Muestra, como su propio nombre indica, una cabecera en la que aparece el nombre de la aplicación, así como el login del usuario para que éste pueda acceder a su perfil.

cal_ocupacion.jsp

Es el calendario en el que se muestra la ocupación de un servicio o subservicio. Presenta en forma de calendario para el mes actual y el siguiente las fechas disponibles y las ocupadas.

Básicamente dibuja dos tablas con los meses pedidos. Al introducir el contenido de cada celda (que será el día correspondiente del mes), la JSP accede, mediante el método *getReserva* de *GestorReservas* al vector de reservas para ese día almacenado en la *Hashtable*. Si este vector contiene alguna reserva la muestra, sino, simplemente se muestra el día en la correspondiente celda.

Para conseguir pasar de un mes a otro mediante flechas (tenedores) se utiliza el método *add* de la clase *GregorianCalendar*.

calendario.jsp

Muy similar al anterior, solo que simplemente muestra las reservas de un usuario para un solo mes. Si es público muestra las reservas realizadas y si es gestor muestra las reservas a su cargo. Es la JSP mostrada en la página de inicio. Al igual que *cal_ocupacion*, se sirve de la clase *GestorReservas*.

dispServ.jsp

Muestra una lista con los servicios y subservicios disponibles entre una fecha de inicio y una de fin, con un icono que permite ver el calendario de ocupación de esos servicios o subservicios. Utiliza las clases *Servicio, Subservicio* y *DispServ*.

errorDispServ.jsp

Maneja los errores que se pueden dar al buscar la disponibilidad de servicios. Estos pueden ser que la fecha introducida por el usuario no es correcta o que el identificador de tipo de servicio no es correcto. Recurre a las excepciones *FechaExcepcion* e *IdTipoServicioExcepcion*.

errorReserva.jsp

Igual que la JSP anterior, esta maneja los errores que se pueden dar al reservar una reserva: el login no existe, se acaba de reservar por otro usuario (para evitar reservas simultáneas) y la base de datos no está disponible. Utiliza las excepciones *ConexiónBDException.java, LoginException.java* y *ReservaExcepcion.java*.

inicio.jsp e inicioG.jsp

Las JSPs de la página de inicio. La única diferencia es que *inicio* muestra la barra lateral para buscar servicios (pues es el inicio de los usuarios públicos), pero *inicioG* no. También muestran la JSP *calendario*.

ocupación.jsp

La JSP de que muestra el calendario de ocupación. Para ello incluye la JSP *cal_ocupacion*. Con los tenedores se permite pasar meses hacia adelante o hacia atrás.

perfil_gestor.jsp y perfil_publico.jsp

*Perfil_gestor p*resenta la información sobre el perfil de los gestores: de que servicios es responsables, las reservas que tiene a su cargo, su foto, etc.... Se sirve de la clase *InfoPerfil*. Por su parte, *perfil_publico* realiza la misma función pero con el perfil de los usuarios públicos.

reserva.jsp

Indica al usuario la conformación de que su reserva ha sido procesada con éxito.

sidebar.jsp

La JSP que presenta la barra lateral para la búsqueda de servicios. Utiliza las clases *TiposServicios* y *dispServBean*.

2regiones.css y 3regiones.css

Las hojas de estilo para las JSPs. *2regiones* determina el estilo para las JSPs de los gestores (que no tienen barra lateral) y *3regiones* determina el estilo de las JSPs del público.

3. Base de datos

La base de datos utilizada en el desarrollo de la aplicación es la proporcionada por la dirección de este proyecto.

Está formada por 5 tablas que son *USUARIOS, TIPO_SERVICIOS, SUBSERVICIOS, SERVICIOS* y *DISPONIBILIDAD_SERVICIOS*.

En la tabla *USUARIOS* se guarda la información de los usuarios: su identificador, nombre, apellidos, e-mail y login. Si el usuario es gestor, en el campo *IdServico* se almacena el identificar de servicio del que es responsable. Si es público, este campo estará vacío.

La tabla *TIPO_SERVICOS* relaciona el identificador de tipo de servicio (*IdTipoServicio*) con su descripción. El tipo de servicio son las diferentes clases de servicios hosteleros: pizzería, restaurante, bar…

En *SERVICIOS* se especifican los diversos negocios de cada servicio hostelero, esto es, los nombres de los diversos negocios. Por ejemplo, el tipo de servicio *pizzería* tiene los servicios *Pizza Hut, Pizza World* y *Domino's Pizza*. También se incluye una pequeña descripción, la url y la capacidad del negocio.

La tabla *SUBSERVICIOS* recoge las diversas ofertas de cada negocio. Por ejemplo *Pizza Hut* tiene los subservicios *Pizza para amigos* y *Pizza guay*. También se indica la capacidad de cada subservicio.

Finalmente, la tabla *DISPONIBILIDAD_SERVICIOS* es la que almacena la información de las reservas, identificando el servicio (nombre de negocio), el subservicio (oferta de ese negocio), la fecha y hora, y el estado de la reserva. Si la reserva está ocupada, también se almacena el login del usuario que realizó la reserva, sino, el campo *Login* estará vacío.

4. Aspectos de diseño

En este apartado se exponen las decisiones más relevantes del diseño, y el porqué de esas decisiones.

Una de las primeras decisiones tomadas fue la de crear una clase especialmente para gestionar las conexiones y desconexiones con la base de datos. Se trata de la clase *ConexionBD*, que posee dos métodos, *CrearConexion* para establecer la conexión y *CerrarConexion* para liberarla. De esta manera se evita tener que escribir todo el código necesario para realizar las conexiones, como registrar el driver. Para poder acceder a la base de datos se utiliza una *environment entry* en el descriptor de despliegue de la aplicación (archivo *web.xml* de la carpeta *WEB-INF*) con la ruta de la base de datos. Con esto se consigue que esta ruta sea fácilmente modificable sin tener que alterar el código, con lo que la aplicación gana en mantenibilidad.

En segundo lugar, es reseñable el uso de la *Hashtable* a la hora de extraer las reservas para ser mostradas en un calendario. Como ya se ha comentado, en esta tabla se asocia a cada fecha un vector de objetos de la clase *reserva* (que almacenan los datos de cada

reserva). Así, a la hora de mostrar al usuario el calendario, se dibuja una tabla donde cada celda es un día. Cada vez que se dibuja una nueva celda, se pide a la clase *GestorReservas* que devuelva el vector de reservas asociado a esa fecha, y este se recorre para mostrar la información de las reservas de cada día. Si este vector está vacío la celda del día correspondiente solo mostrará el número del día. El problema es esta forma de obtener las reservas puede resultar ineficiente, sobre todo si hay pocas reservas en un mes. Para cada día se accede a la *Hashtable* y se comprueba si hay alguna reserva, con lo que puede ser que accedamos muchas veces a la tabla y no obtengamos ninguna información (no hay reservas). Aún así, esta forma de proceder nos parece correcta por ser sencilla de entender, programar y modificar en caso de necesidad.

Otro aspecto de diseño importante es la estructuración de las clases. Se ha intentado que, además de los servlets, se disponga de dos tipos de clases: unas para extraer o actualizar datos de la base de datos y otras para almacenar esos datos y acceder a ellos desde las JSPs. Este diseño hace a la aplicación más modular, de manera que sea más sencillo detectar y corregir errores.

Finalmente, se ha intentado que los errores más comunes que puedan surgir se manejen a través de excepciones propias para poder mostrárselos al usuario de una manera más amigable. De esta manera se consigue un entorno más amable, evitando mensajes de error que puedan resultar impactantes para un usuario medio.

5. Referencias

[1] E. Villoslada de la Torre. *"Web de Reservas. Especificación de Requisitos"*. Desarrollo de aplicaciones orientadas a Internet. ETSIT UVA. Telefónica I+D. Septiembre de 2010.