TRABAJO FINAL DE CARRERA

Calculadora de Rutas Turísticas

Ezequiel Foncubierta Estévez
Ingeniero Técnico en Informática de Sistemas

Índice de contenido

1. INTRODUCCIÓN

El presente documento recoge la memoria del **Trabajo Final de Carrera** realizado por *Ezequiel Foncubierta Estévez* para obtener la titulación de *Ingeniería Técnica en Informática de Sistemas* por la *Universitat Oberta de Catalunya.* El contenido que aquí se expone ha sido desarrollado durante el primer semestre del curso 2008/2009, donde se han desarrollado tres pruebas de evaluación continua cuyos resultados se han utilizado para escribir esta memoria.

1.1 Justificación del proyecto

La idea del proyecto, parte de la necesidad de crear una red social que permita a los usuarios crear puntos de interés y, que otros usuarios, puedan explotar la información para la obtención de los puntos más interesantes y más cercanos a una ruta preestablecida. Es decir, si un usuario desea ir desde Cádiz hasta Málaga, pasando por Sevilla, ¿Qué puntos de interés podrá visitar?

Todo esto, supervisado por moderadores que garanticen la fiabilidad de la información.

1.2 Objetivos del trabajo

El objetivo del trabajo es definir y desarrollar los mecanismos necesarios para:

- ➔ Gestionar puntos de interés
- ➔ Gestionar los parámetros de clasificación de los puntos de interés
- ➔ Moderar y atender los avisos de los usuarios
- ➔ Calcular rutas basadas en los criterios del usuario

El proyecto debe cumplir además que:

- ➔ Sea una arquitectura fácilmente escalable y orientada a componentes
- ➔ Sea independiente de la plataforma, arquitectura y SGBD
- ➔ Use tecnologías 100% Opensource, exceptuando las tareas de diseño

No es objetivo de este proyecto:

- ➔ Definir un algoritmo de calculo de rutas, ya que dada su complejidad abarcaría un único proyecto para su implementarlo. Para este caso, se diseñará un motor de rutas básico para cubrir la funcionalidad, dejando abierta la posibilidad de inyectar, a través de Spring, cualquier motor de cálculo de rutas.
- ➔ Implementar todas las pantallas HTML. **Esto no quiere decir** que los Actions de Struts y que la capa de servicio a datos no estén completamente desarrollados. El motivo por el que no se desarrollan todas las pantallas es por el corto periodo de tiempo para el desarrollo del proyecto, siendo otorgada mayor importancia al desarrollo de los servicios de acceso a datos, así como a toda la parte de desarrollo Java, dejando para el final el desarrollo de las pantallas HTML.

1.3 Planificación del proyecto

En un principio se comenzó por el desarrollo del modelo de datos y la capa de negocio, que es la base para el acceso a la información utilizada los Portlets. Posteriormente se desarrollaron los cuatros Portlets, o componentes, implicados en el proyecto.

Con todo desarrollado, se realizaron las tareas de integración y pruebas, finalizando con al documentación de la presente memoria.

Los hitos del desarrollo en orden cronológico han sido:

28/10/2008: Disponible el modelo de datos y capa de negocio

11/11/2008: Creado el Portlet de gestión de puntos de interés

25/11/2008: Creado el Portlet de administración de puntos de interés

09/12/2008: Creado el Portlet de moderación de avisos

23/12/2008: Creada la Calculadora de itinerarios

14/01/2009: Finalizada la documentación y entrega del proyecto

	Nombre de tarea	Duración	Comienzo	Fin	Pred
1	**Modelo de datos**	**20 días**	**mié 01/10/08**	**mar 28/10/08**	
2	Análisis y diseño	10 días	mié 01/10/08	mar 14/10/08	
3	Desarrollo SQL	3 días	mié 15/10/08	vie 17/10/08	2
4	Desarrollo capa de negocio	7 días	lun 20/10/08	mar 28/10/08	3
5	**Portlets**	**40 días**	**mié 29/10/08**	**mar 23/12/08**	**1**
6	**Gestión de puntos de interés**	**10 días**	**mié 29/10/08**	**mar 11/11/08**	
7	Análisis y diseño	5 días	mié 29/10/08	mar 04/11/08	
8	Desarrollo	10 días	mié 29/10/08	mar 11/11/08	
9	**Administración de puntos de**	**10 días**	**mié 12/11/08**	**mar 25/11/08**	**6**
10	Análisis y diseño	5 días	mié 12/11/08	mar 18/11/08	
11	Desarrollo	10 días	mié 12/11/08	mar 25/11/08	
12	**Moderador de avisos**	**10 días**	**mié 26/11/08**	**mar 09/12/08**	**9**
13	Análisis y diseño	5 días	mié 26/11/08	mar 02/12/08	
14	Desarrollo	10 días	mié 26/11/08	mar 09/12/08	
15	**Calculadora de itinerarios**	**10 días**	**mié 10/12/08**	**mar 23/12/08**	**12**
16	Análisis y diseño	5 días	mié 10/12/08	mar 16/12/08	
17	Desarrollo	10 días	mié 10/12/08	mar 23/12/08	
18	**Integración y pruebas**	**2 días**	**mié 24/12/08**	**jue 25/12/08**	**15**
19	**Documentación TFC**	**14 días**	**vie 26/12/08**	**mié 14/01/09**	**18**

Ilustración 1: Planificación de tareas

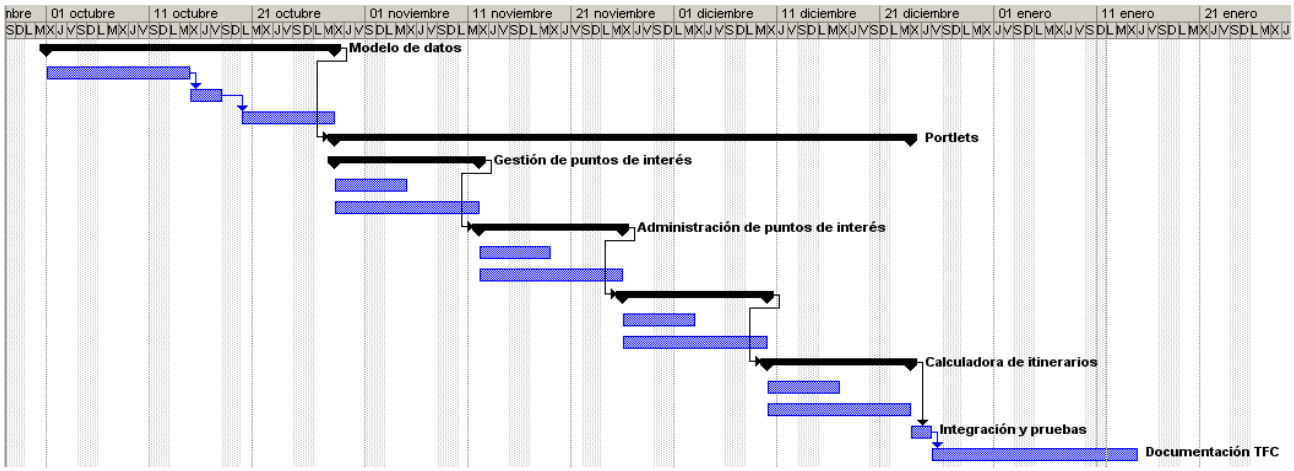

Ilustración 2: Cronología de tareas

1.4 Productos obtenidos

Como resultado, el proyecto generará un único producto, además de la documentación entregada a lo largo de su desarrollo. El producto resultante es un proyecto Java que genera una distribución de la aplicación.

2. ESPECIFICACIÓN Y REQUERIMIENTOS

2.1 Información inicial

La plataforma de desarrollo para implementar la solución es Java JEE, que se ha convertido en el estándar de facto para el desarrollo de aplicaciones corporativas en el entorno Web.

Como base tecnológica, además de Java JEE, se han utilizado los siguientes frameworks:

- **Struts**: Permite crear aplicaciones Web usando un diseño de arquitectura de tres capas MVC (*Model View Controller*), la cual permite separar la gestión de los datos de la presentación.

- **Spring**: Implementa un contenedor *Inversion of Control (IoC)*, que gestiona y maneja beans, instanciándolos, configurándolos y ensamblando sus dependencias haciendo uso del conocido patrón *Dependency Injection*, el cual se explicará con mayor profundidad en las siguientes secciones.

- **Hibernate**: Motor de persistencia de los objetos en base de datos que permite, entre otras cosas, realizar operaciones de lectura/escritura en la base de datos sin necesidad de conocer cual es el sistema gestor de base de datos que está por detrás.

2.2 Requisitos Tecnológicos

Antes de entrar a describir la estructura y diseño del proyecto, daremos un pequeño repaso a cada una de las tecnologías que se han utilizado para desarrollar el proyecto, sin entrar a detallar todas las características de cada una de ellas. Si se desea profundizar, al final del documento se ha creado una entrada bibliográfica clasificada por cada una de las tecnologías utilizadas.

2.2.1 Java JEE

El lenguaje Java nace de la mano de Sun Microsystems a principio de los años 90 con el objetivo de ser un lenguaje orientado 100% a objetos, eliminando el desarrollo a bajo nivel introducido por lenguajes estructurados tipo C, C++ o Eiffel. El lenguaje se desarrolla para cumplir, entre otras cosas, las siguientes características:

- Ser totalmente orientado a objetos: El lenguaje debe dar soporte a *Clases* y *Objetos*, además de soportar los conceptos *Interfaz, Herencia* y *Polimorfismo*.

- Independencia de la plataforma: Los programas desarrollados tienen que poder ser ejecutados en cualquier plataforma, y arquitectura hardware, a través de una máquina virtual.

- Recolección de basura: Tiene que existir un mecanismo para liberar los espacios de memoria reservados para alojar los objetos, problema muy común de C y C++ al hacer operaciones de *Memory Allocation*.

En el auge de las *puntocom*, por el año 1998, Sun Microsystems decide incluir en la plataforma un conjunto de herramientas y librerías orientadas al desarrollo de aplicaciones empresariales. Con la publicación de Java J2SE 1.3 se publica la primera versión de Java J2EE. Dada la versatilidad y

potencia del lenguaje, Java comienza a abrirse paso en el mercado y muchos equipos de desarrollo comienzan a desarrollar Frameworks, cada vez más potentes, que son acogidos con gran expectación.

A partir de este momento, Sun Microsystems apuesta fuertemente por Java JEE llegando hasta la publicación de Java JEE 5 en el año 2006, versión que se ha utilizado en el desarrollo del proyecto. La plataforma JEE define las siguientes APIs generales:

- **javax.ejb.***: Define la API *Enterprise JavaBeans*, que son objetos distribuidos para el manejo de transacciones, persistencia, etc...
- **javax.naming.***: Define la API de *Java Naming* y *Directory Interface (JDNI)*.
- **java.sql y javax.sql**.*: Define la API *Java DataBase Connectivity* que modela los tipos de datos usados en Bases de Datos, además de definir los conectors básicos para que sean implementados por los drivers de conexión a cada tipo de base de datos
- **java.transaction.***: Define la API *Java Transaction (JTA)*, que incluye una serie de interfaces para el desarrollo de aplicaciones distribuidas
- **javax.xml.***: Define la API *Java Api for Xml Processing (JAXP)*, que sirve para la manipulación de documentos XML
- **javax.jms.***: Define la API *Java Message Service (JMS)* a través de la cual se pueden gestionar colas de mensajes

2.2.2 JSR-168 (Portlet)

La especificación JSR-168 define un conjunto de APIs para el desarrollo de componentes Webs, además de definir los mecanismos para la interacción de los portales con dichos componentes. La especificación define:

- **Portlet**: Componente que produce un fragmento de código HTML, utilizado por un portal para componer una página Web. Los Portlets son aplicaciones Webs JEE y tienen su propio contexto de ejecución. Cuando se desarrolla un Portlet, realmente se está desarrollando una aplicación Web, ofreciendo, por tanto, la posibilidad de utilizar cualquier tecnología Java por detrás de la especificación JSR-168.
- **Contenedor de Portlets**: Los portales que quieran incluir Portlets como tecnología han de implementar un contenedor que cumpla con la especificación JSR-168.

La especificación define las siguientes funcionalidades para los Portlets:

- **Almacenamiento persistente**: Los Portlets disponen de un objeto *PortletPreferences* para almacenar las preferencias del usuario, siendo labor del contenedor de Portlets almacenar la información en la base de datos.
- **Procesamiento de solicitudes**: Los Portlets son aplicaciones Webs, con su propio contexto de ejecución. Cuando un usuario realice una petición, el Portal solicitará al contenedor de Portlets que realice una petición a cada uno de los Portlets implicados en la página solicitada por el

usuario. Será por tanto, el contendor de Portlets el que realizará solicitudes HTTP a los Portlets para obtener el fragmento de código HTML que será entregado al Portal.

➔ **Modos de visualización**: Los Portlets pueden visualizar información de tres tipos:

- VIEW: Sería el comportamiento normal y mostraría el fragmento HTML correspondiente a la acción que esté realizando el Portlet. La vista VIEW es obligatoria.
- EDIT: Muestra un formulario para editar la configuración de los Portlets.
- HELP: Muestra una página con ayuda sobre el Portlet.

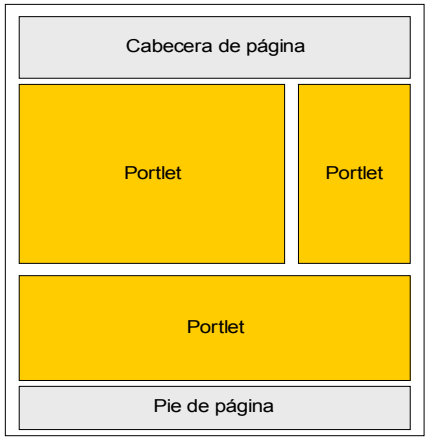

Ilustración 3: Estructura del portal

➔ **Estado de la ventana**: Un portlet en un portal, como se puede ver en la *Ilustración 3*, es como una ventana en un sistema operativo y puede tener tres estados básicos.

- NORMAL: El Portlet ocupa su espacio asignado en la página.
- MINIMIZADO: El Portlet solo muestra la barra de título sin mostrar sus contenido
- MAXIMIZADO: El Portlet se muestra ocupando toda la página

➔ **Atributos de usuario**: Los Portlets tienen la capacidad de acceder a la información del usuario gestionada por el Portal.

El objetivo de la tecnología es que los componentes Web puedan ser totalmente portables, de tal forma que puedan ser utilizados por otros portales sin necesidad de escribir ninguna linea de código para adaptarlo. La especificación Portlet ha sido incluida por Sun como parte de la plataforma JEE.

2.2.3 Struts

El framework Struts se basa en el patrón Modelo-Vista-Controlador (MVC). Nace de la mano de Apache Foundation, pero en su última versión 2.X se basa en el recién liberado framework propietario WebWork.

Se trata de un framework orientado a acciones, donde cada una de éstas realiza, o debería de realizar, una única tarea (Guardar un dato, Visualizar un listado, etc...).

2.2.4 Spring

Fue creado en alrededor del año 2002 por Rod Jonhson, aunque su primera versión oficial se publicó en el año 2004. La función del framework es crear un modelo de desarrollo con las mejores prácticas, facilitando su integración con la tecnología JEE.

Con Spring nace el concepto *Inversion of Control (IoC)*, cuyo mecanismo es inverso a la programación tradicional, donde cada componente instancia y llama a los procedimientos

necesarios. Con *IoC*, un contenedor es el encargado de instanciar y ensamblar cada componente con los objetos que éste requiera. El principio de esta técnica vienen del patrón *Dependency Inyection.* A esta técnica también se le conoce, coloquialmente, como el *Principio de Hollywood*, una metodología de diseño de software, cuyo transfondo es *no nos llames a nosotros, nosotros te llamaremos a ti*.

Spring comienza a ganar adeptos, y el framework crece de una forma exponencial con multitud de colaboradores, dotándolo con herramientas de persistencia, Web, distribuidas, etc...

2.2.5 Hibernate

Es una herramienta de persistencia, que mapea objetos Java con tablas de bases de datos. El objetivo es abstraer a la aplicación de las llamadas a la base de datos, mejorando la portabilidad de la aplicación al ser delegada las llamadas a base de datos en Hibernate, siendo totalmente transparente el gestor de base de datos que se está utilizando.

Hibernate se basa en el uso de dialectos para convertir los tipos de datos Java a tipos de datos conocidos por la base de datos. De esta forma, cambiando el dialecto a utilizar, y la conexión a la base de datos, basta para entenderse con cualquier tipo de base de datos. Siempre y cuando exista el dialecto a utilizar para dicha base de datos.

2.3 Requisitos Funcionales

El proyecto se divide en varios componentes, o Portlets, cada uno de los cuales ha de implementar un conjunto de funcionalidades. A continuación se detallan cuales son las funcionalidades que tienen que cubrir cada uno de ellos.

2.3.1 Portlet: Points Manager

Ofrece las herramientas necesarias para la gestión de puntos de interés, donde se podrán crear o editar los puntos, creados o modificados por el usuario que ejecuta el Portlet. Para poder utilizar el Portlet, es necesario que el usuario esté autenticado en el sistema, porque tiene que quedar constancia de quien y cuando se ha creado o modificado el punto de interés.

Debe de implementar las siguientes funcionalidades:

- ➔ Listado de puntos de interés que hayan sido creados, o modificados, por el usuario que carga el listado
- ➔ Alta de nuevo punto de interés
- ➔ Edición de un punto de interés
- ➔ Eliminación de un punto de interés

2.3.2 Portlet: Points Moderator

Cada vez que se crea, o se modifica, un punto, se crea un aviso para que el usuario moderador revise la información y la publique si lo ve conveniente. A través de este Portlet, los moderadores

podrán ver los avisos que se han ido generando y poder actuar en consecuencia. Estos avisos pueden tener, en principio, tres estados:

- **open**: El aviso ha sido creado y está esperando la atención del moderador
- **closed**: El moderador ya ha atendido el aviso y lo da por cerrado
- **declined**: El moderador rechaza el aviso por no estar de acuerdo con el aviso del usuario

Las funcionalidades que debe implementar este Portlet son:

- Listado de todos los avisos generados
- Visualización de un determinado aviso
- Si el aviso es sobre un punto de interés, debe permitirse habilitar el punto de interés al que hace referencia el aviso

2.3.3 Portlet: Points Admin

Los usuarios administradores necesitan gestionar el entorno de ejecución de la aplicación. Para el alcance del proyecto, únicamente podrán gestionar los estados y las categorías que se le pueden dar a los avisos. Debe de implementar las siguientes funcionalidades:

- Listado de todas las categorías
- Alta de una categoría
- Edición de una categoría
- Eliminación de una categoría
- Listado de todos los estados de aviso
- Alta de un estado de aviso
- Edición de un estado de aviso
- Eliminación de un estado de aviso

2.3.4 Portlet: Route Calculator

Todos los puntos publicados en el sistema son explotados por la calculadora de rutas, componente a través del cual el usuario indicará un punto de inicio y un punto de fin para obtener una ruta con puntos los puntos de interés más cercanos. Cuando el usuario vaya a calcular una ruta tendrá que especificar los siguientes criterios:

- **Punto de inicio y punto final**: Son los puntos salida y llegada de la ruta. Estos parámetros son obligatorios para el cálculo de una ruta
- **Puntos intermedio**: Conjunto de puntos por los que desea pasar

➔ **Tipos de puntos**: Tipos de puntos que se quiere encontrar en la ruta. De esta forma el usuario podrá obtener, únicamente, los puntos de su interés

Las funcionalidades que tiene que implementar este Portlet son:

➔ Calculo de una ruta turística partiendo de los criterios definidos por el usuario, que en principio son: Punto de inicio, punto final y categoría de los puntos a mostrar

➔ Visualización de un punto de interés

➔ Comentar un punto de interés

➔ Evaluar un punto de interés

2.4 Arquitectura del sistema

El objetivo del proyecto es la construcción de Portlets para la gestión, administración de puntos de interés, mas un Portlet para la explotación en forma de calculadora de rutas. Es necesario, por tanto y según la especificación **JSR-168**, un portal ó contenedor de Portlets. La herramienta más común utilizada para desarrollar Portlets es **Apache Pluto**, aunque debido a su simplicidad y carencia en la gestión de usuarios se ha decidido utilizar la herramienta opensource **Liferay Portal**.

Como se puede ver en el siguiente diagrama, cuando el usuario inicia una transacción con el Portal a través del protocolo HTTP, entran en acción varios componentes:

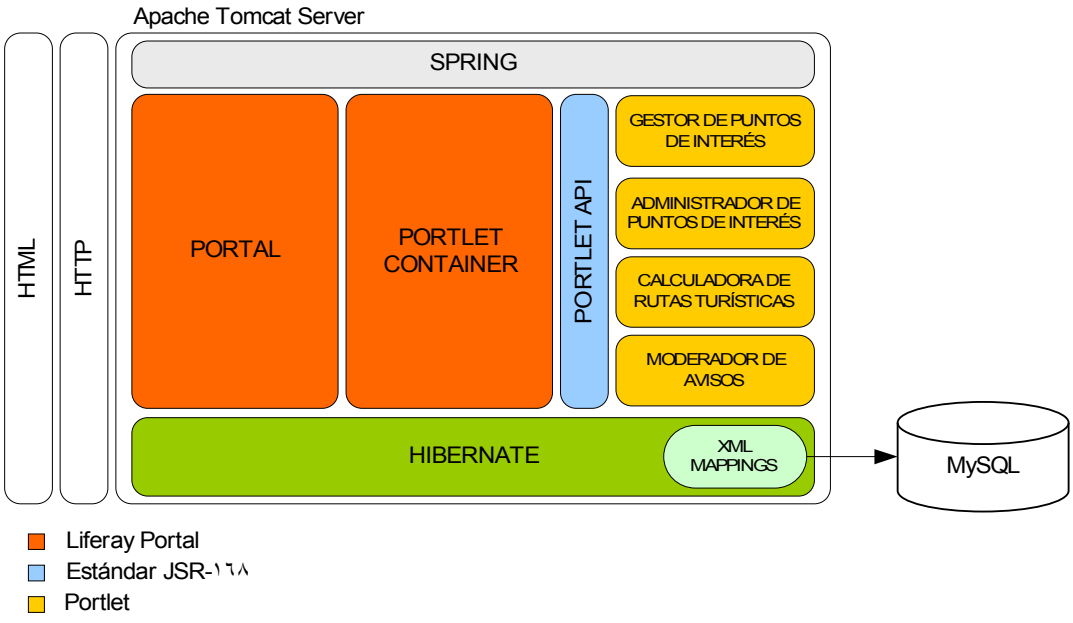

Ilustración 4: Arquitectura del sistema

➔ Portal: Se encarga de gestionar la petición del usuario, configurar el contexto de ejecución del resto de componentes y pasar la llamada al componente *Portlet Container* para obtener el código HTML que se mostrará al usuario

➔ Portlet Container: Continua con la petición del usuario y, dependiendo de la página solicitada,

invoca el conjunto de Portlets definidos en dicha página. Para una correcta comunicación se utiliza la API estándar JSR-168, de tal forma que los Portlets puedan ser creados por terceros e independientemente del Portal que los ejecute

- Conjunto de Portlets: Son los componentes que se muestran en el Portal y el objetivo del presente proyecto. Implementan la API estándar JSR-168 y actúan como aplicaciones independientes del portal, aunque en lugar de generar páginas completas, generan fragmentos de código HTML para que sean embebidos en el portal

Además de estos componentes, se pueden ver otros servicios horizontales como Spring e Hibernate utilizados como base para la inyección de dependencia y persistencia de datos respectivamente.

Lógicamente existen más tecnologías horizontales como Struts, Xerces, etc... utilizadas por Liferay aunque no es objetivo del presente documento el enunciar cada una de ellas, ya que no están implicadas en el desarrollo de los Portlets del proyecto.

3. ANÁLISIS

3.1 Diagrama de casos de uso

Los casos que tiene que resolver los Portlets del proyecto son los marcados en amarillo. Los marcados en rojo son casos de uso importantes que están resueltos por la herramienta ***Liferay Portal***.

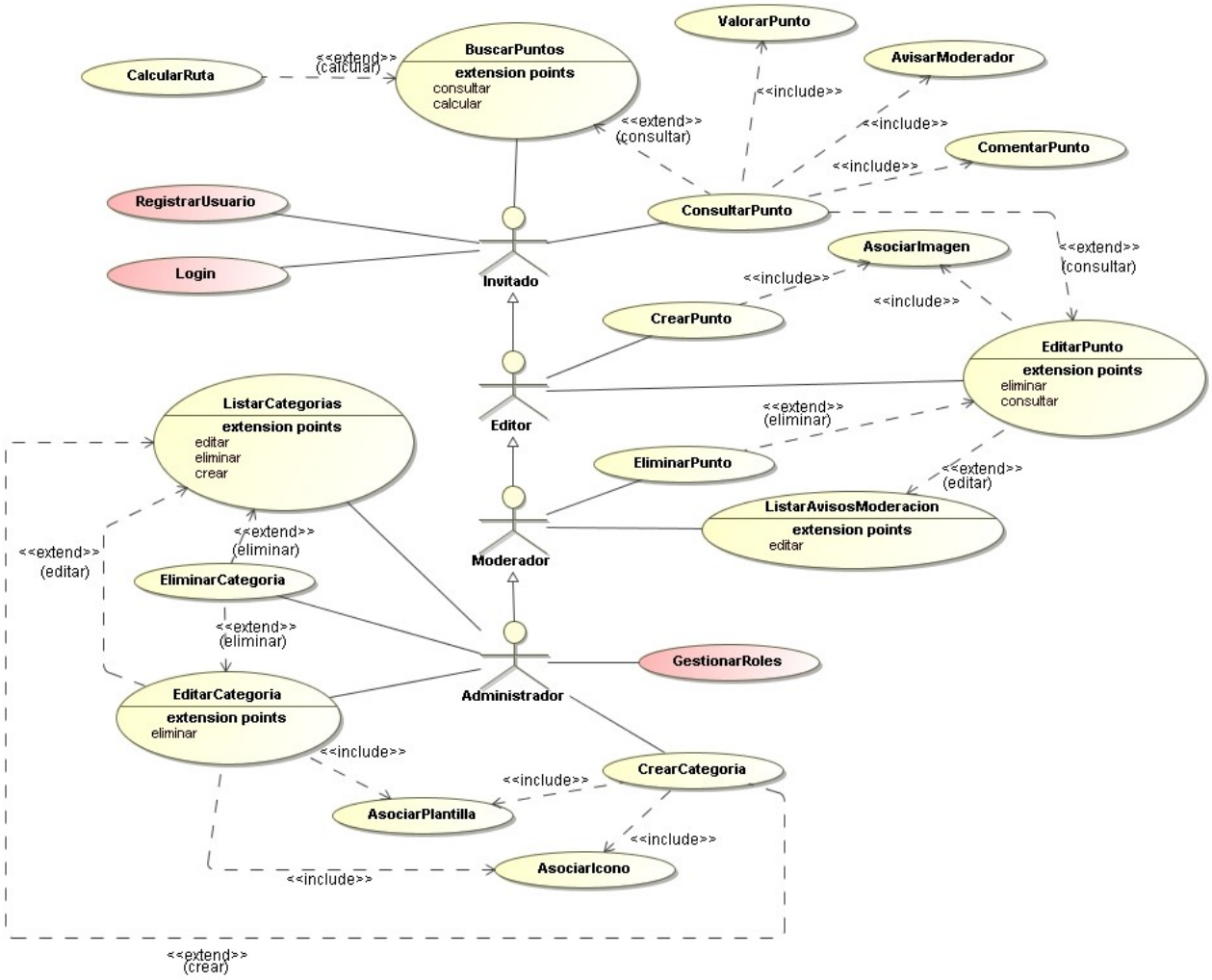

Diagrama 1: Principales casos de uso

A continuación se da una descripción de cada caso de uso:

- ***BuscarPuntos***: El usuario hace uso del portlet *Calculadora de Rutas* a través del cual, y bajo los filtros de búsqueda deseados, obtendrá un conjunto de puntos de interés como resultado de su consulta. Con el resultado obtenido, el usuario podrá consultar cada uno de los puntos encontrados.

- ***ConsultarPunto***: El usuario puede ver los detalles de un determinado punto (imagen, descripción, posición, etc...). Además podrá:

- ***ValorarPunto***: Dar una valoración al punto de interés que está consultado.
- ***AvisarModerador***: Si se encuentran incoherencias en la información que se está mostrando, ya sea por una errata textual o por que la información que se presenta no coincide con el punto de interés, el usuario puede poner en aviso a un moderador para que revise, y corrija si fuese necesario, la información.
- ***ComentarPunto***: Los usuario podrán realizar comentarios sobre los puntos de interés, de tal forma que puedan complementar la información mostrada o valorar textualmente su opinión sobre el punto de interés.

➔ ***CrearPunto***: Los usuarios con poder de edición, podrán crear un nuevo punto de interés de la categoría que deseen. Dicho punto creado pasará por moderación antes de ser público.

➔ ***EditarPunto***: Los usuarios podrán realizar modificaciones sobre cualquier punto existente a modo de moderación o aplicación de la información, aunque dicha acción será igualmente supervisada por el moderador.

➔ ***EliminarPunto***: Si existe un punto que no presenta la información correcta, o presenta discrepancia entre los usuarios, podrá ser eliminado del sistema por el moderador.

➔ ***ListarAvisosModeracion***: Los usuarios moderadores podrán consultar los avisos de moderación por parte del resto de usuarios, de tal forma que puedan editar directamente dicho punto.

➔ ***ListarCategorias***: El usuario administrador podrá acceder a la lista de categorías que clasifican los puntos de interés, de tal forma que pueda:

- ***EditarCategoria***: Modificar los datos de una determinada categoría (plantilla, icono, descripción).
- ***EliminarCategoria***: Dar de baja una determinada categoría.
- ***CrearCategoria***: Crear una nueva categoría asignándole una determinada plantilla y un icono de representación.

Los actores implicados en el manejo de los Portlets son:

➔ **invitado**: Persona que utilizará la aplicación para realizar consultas a través de la calculadora de rutas. Con los resultados de la búsqueda el usuario podrá consultar cada uno de los puntos de interés encontrado y realizar valoraciones sobre el mismo, además de comentarlo o avisar de incidencias encontradas

➔ **editor**: Cualquier persona registrada en el sistema podrá crear nuevos, o editar, puntos de interés bajo la supervisión de un moderador

➔ **moderador**: Persona encargada de supervisar los puntos que se crean, conservando así la integridad de los datos que verá el usuario. Además, podrán recibir avisos de moderación por parte de los usuario que hayan encontrado erratas en la información

➔ **administrador**: Persona que supervisa todo el sistema, podrá crear nuevas categorías para los puntos de interés, gestionar los roles de los usuarios y en general, administrar todo el sistema

3.2 Modelo de datos

El modelo del dominio presentado son objetos pojo que modelan los datos necesarios para la gestión de la información de la calculadora de rutas.

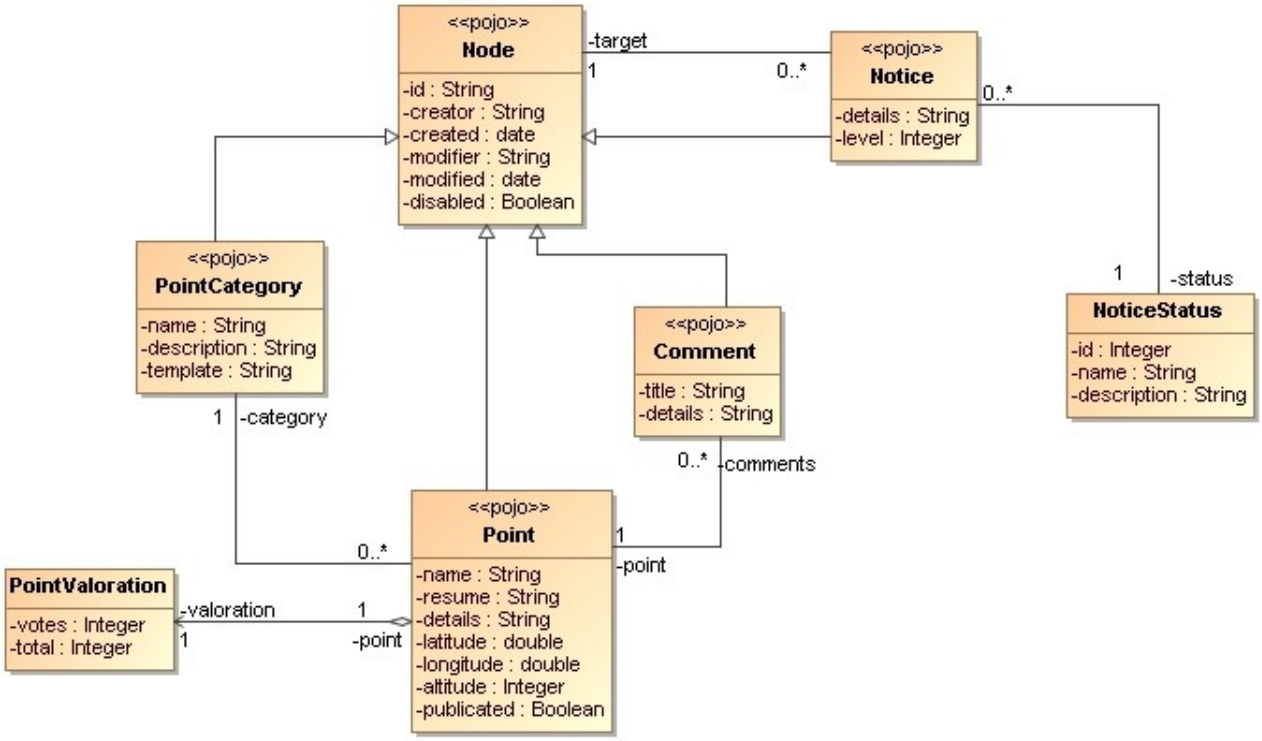

Diagrama 2: Modelo de datos

- ***Node***: Tipo de dato principal que incluye información para auditar los datos:
  - **id**: Identificador del contenido
  - **creator**: Identificador del usuario que ha creado el contenido
  - **created**: Fecha de creación
  - **modifier**: Identificador del usuario que ha modificado el contenido
  - **modified**: Fecha de modificación
  - **disabled**: Para realizar borrados lógicos de un dato
- ***PointCategory***: Nodo que gestiona las categorías de se le pueden asignar a los puntos de interés
  - **name**: Nombre de la categoría
  - **description**: Descripción sobre la categoría
  - **template**: Plantilla que se aplica al visualizar el punto de interés con dicha categoría
- ***Point***: Nodo que gestiona los puntos de interés

  - **name**: Nombre, o título, del punto de interés
  - **resume**: Texto corto que resume el punto de interés
  - **details**: Texto extendido con todos los detalles sobre el punto de interés
  - **latitude y longitude**: Coordenadas geográficas del punto de interés
  - **altitude**: Altura sobre el nivel del mar del punto de interés
  - **publicated**: Indica si el punto está publicado, y visible en el portal, o no
  - **valoration**: Valoración que hacen los usuarios en un punto de interés
- ***PointValoration***: Modela la valoración que hacen los usuarios
  - **point**: Punto de interés sobre el que se realiza la valoración
  - **votes**: Número de votos totales que se han realizado
  - **total**: Suma total de la votación (media = total / votes)
- ***Comment***: Nodo que gestiona los comentarios
  - **title**: Titulo del comentario
  - **details**: Texto del comentario
- ***Notice***: Nodo que gestiona los avisos
  - **details**: Texto del aviso
  - **level**: Indica el nivel crítico del aviso (1:Crítico, 2:Alto, 3:Normal, 4:Bajo)
  - **status**: Estado en el que se encuentra el aviso (Abierto, Cerrado, etc...)
- ***NoticeStatus***: Modela los estados de los avisos
  - **id**: Identificador del estado
  - **name**: Nombre del estado
  - **description**: Descripción del estado

4. DISEÑO TÉCNICO

4.1 Patrones de diseño

4.1.1 Model-View-Controller (MVC)

MVC no es realmente un patrón de diseño sino una buena práctica en el diseño de arquitectura de aplicaciones. Consiste en dividir la aplicación en tres capas, separando la interfaz de usuario de la gestión de los datos a través de una lógica de control. Cada capa define una función bien definida:

- **Model** (Modelo)**:** Define los mecanismos para la representación de la información y, generalmente, de toda la lógica de negocio que cubre la aplicación, asegurando la integridad y la fiabilidad de la información. Comúnmente, el modelo suele ser una capa de servicios sobre el Sistema de Gestión de Bases de Datos.

- **View** (Vista): Define las pantallas que verá el usuario para interactuar con la información gestionada por el modelo.

- **Controller** (Controlador): Define la lógica que atiende los eventos, o peticiones, generados por el usuario, a través de la vista, realizando las operaciones oportunas en la capa de servicios definida en el modelo.

4.1.2 Data Access Object (DAO)

Es un patrón introducido en el desarrollo de aplicaciones J2EE y consisten en definir una capa de acceso a los datos, de tal forma que la aplicación no tenga por que conocer el destino de los datos, necesitando conocer únicamente las operaciones definidas para cada tipo de dato.

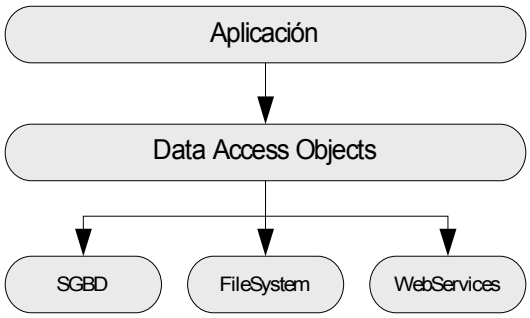

Ilustración 5: Patrón DAO

La forma usual de implementar este patrón, es definir las interfaces DAO que son las que utilizarán las clases que acceden a datos para conocer cuales son las operaciones que pueden realizar. Por otro lado, se implementan estas interfaces con la lógica de acceso a datos correspondiente:

```
public interface PersonaDao {
	public Persona getPersona (String dni);
	public List<Persona> getPersonas ();
}

public class PersonaDaoImpl implements PersonaDao {
	public Persona getPersona (String dni) {
		/** Lógica correspondiente de acceso a datos **/
	}
}
```

Siguiendo el ejemplo anterior, la aplicación solo necesita conocer la interfaz PersonaDao para definir sus operaciones de acceso a datos.

4.1.3 Dependency Injection (DI)

El término Inyección de dependencias fue introducido por el creado de Spring, Martin Fowler. Consiste en resolver las dependencias, o los atributos, de cada clase generando todos los objetos al inicio de la aplicación y luego inyectárselo a los objetos que lo necesiten a través de su constructor o sus métodos set. Con esto se mejora la eficiencia del sistema al tener todos los objetos instanciados en una factoría que será compartida por todos los usuarios.

Siguiendo el ejemplo del punto anterior:

```
public class ManejadorUsuarios {
      private PersonaDao personaDao;

      public void setPersonaDao (PersonaDao personaDao) {
            this.personaDao = personaDao;
      }
}
```

La clase *ManejadorUsuarios* utiliza un objeto *personaDao* del que desconoce su implementación (recordemos que *PersonaDao* es una interfaz) y que por tanto no instancia. En su lugar, espera que le asignen un objeto de este tipo a través de un método set del mismo nombre.

A lo largo del documento se hace referencia a "*método que inyecta el servicio*". Esto quiere decir, que ese objeto será utilizado a través de Spring y cuyas dependencias, también gestionadas por Spring, le serán inyectadas al iniciarse la aplicación.

4.2 Servicios de acceso a datos

Siguiendo el patrón DAO, mencionado en el punto anterior, se ha diseñado la siguiente capa servicios de acceso a datos.

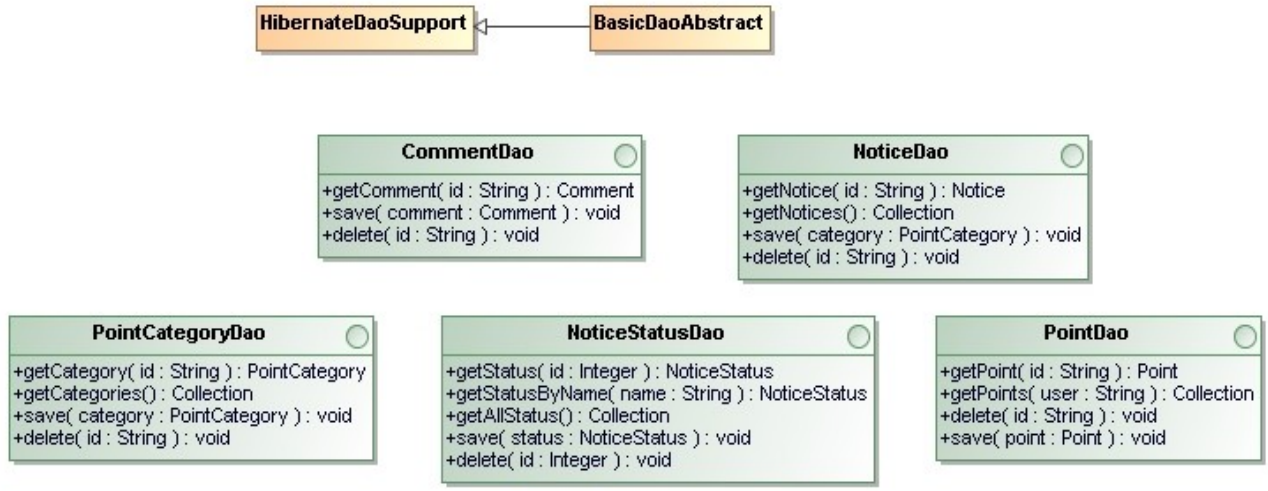

Diagrama 3: Servicios de acceso a datos

Para cada tipo de dato que se va a manejar, se ha diseñado una interfaz con todas las operaciones de entrada/salida que podrá realizar.

➔ ***PointCategoryDao***: Define las operaciones de acceso a datos para las categorías

- **getCategory**: Devuelve una categoría dado su identificador. Si no existe devuelve null
- **getCategories**: Devuelve un listado con todas las categorías
- **save**: Guarda un objeto de tipo categoría
- **delete**: Elimina un objeto de tipo categoría dado su identificador

➔ ***CommentDao:*** Define las operaciones de acceso a datos para los comentarios

- **getComments**: Devuelve todos los comentarios de un punto de interés dado su identificador. Si el punto no existe, devuelve null
- **save**: Guarda un objeto de tipo comentario
- **delete**: Elimina un objeto de tipo comentario dado su identificador

➔ ***NoticeStatusDao:*** Define las operaciones de acceso a datos para los estados de avisos

- **getStatus**: Devuelve un objeto estado dado su identificador. Si no existe devuelve null
- **getStatusByName**: Devuelve un objeto estado dado su nombre. Si no existe devuelve null
- **getAllStatus**: Devuelve todos los estados
- **save**: Guarda un objeto de tipo estado
- **delete**: Elimina un objeto de tipo estado dado su identificador

➔ ***NoticeDao***: Define las operaciones de acceso a datos para los avisos

- **getNotice**: Devuelve un objeto aviso dado su identificador. Si no existe devuelve null
- **getNotices**: Devuelve un listado con todos los avisos
- **save**: Guarda un objeto de tipo aviso
- **delete**: Elimina un objeto de tipo aviso dado su identificador

➔ ***PointDao***: Define las operaciones de acceso a datos para los puntos de interés

- **getPoint**: Devuelve un objeto punto de interés dado su identificador. Si no existe devuelve null
- **getPoints**: Devuelve un listado con todos los puntos de interés de un determinado usuario
- **save**: Guarda un objeto de tipo punto de interés
- **delete**: Elimina un objeto de tipo punto de interés dado su identificador

Además de definir las operaciones, se ha definido una clase abstracta de la que heredan las implementaciones de las interfaces de la capa de servicios de acceso a datos.

La clase abstracta *BasicDaoAbstract* hereda a su vez de *HibernateDaoSupport*. Esta última clase se distribuye con el framework Spring y, además de definir los métodos necesarios para inyectarle el servicio *sessionFactory* (como se verá en el capítulo 5), implementa un método *getHibernateTemplate,* que devuelve un objeto *HibernateTemplate,* a través del cual se realizarán todas las operaciones de lectura/escritura en la base de datos.

4.3 Portlets

Los Portlets que se definen son todos Struts Portlets. Esto quiere decir, que para el desarrollode cada Portlet es similar al desarrollo de una aplicación Struts con el inconveniente de que la aplicación se ejecute a través de una implementación de un Portlet estándar, según lo define la especificación JSR-168.

En la versión 2 de Struts se da soporte a la especificación JSR-168, proporcionando la clase *org.apache.struts2.portlet.dispatcher.Jsr168Dispatcher.* Esta clase hereda de *javax.portlet.GenericPortlet,* que es la clase que ofrece la API definida en la especificación JSR-168 para definir un Portlet estándar.

Del dispatcher JSR-168 definido por Struts, heredan cada uno de los Portlets de la calculadora de rutas, tal y como se puede apreciar en el siguiente diagrama, consiguiendo así que las aplicaciones Struts desarrolladas para cada Portlet se ejecuten a través de una implementación Portlet válida.

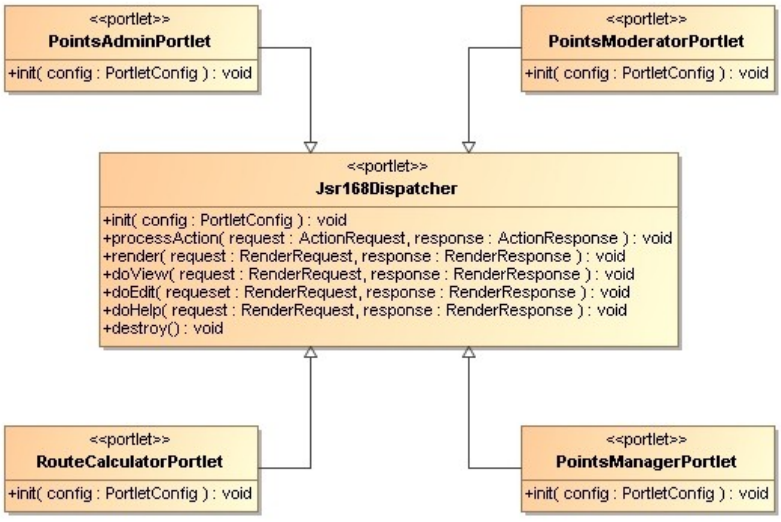

Diagrama 4: Portlets

El dispatcher, implementa los siguientes métodos definidos en la clase *GenericPortlet*:

➔ **init**: Método que se ejecuta al iniciar el Portlet. Su función es leer los parámetros de configuración definidos en el fichero de descripción de Portlets (*portlet.xml*). Los parámetros de configuración que puede recibir son:

- **viewNamespace**: Define el namespace de Struts 2 donde se encuentra la acción que se ejecutará por defecto para la vista VIEW.

- **defaultViewAction**: Nombre del Action por defecto para la vista VIEW del Portlet.
- **editNamespace**: Define el namespace de Struts 2 donde se encuentra la acción que se ejecutará por defecto para la vista EDIT.
- **defaultEditAction**: Nombre del Action por defecto para la vista EDIT del Portlet.

➔ **processAction**: Ejecuta la acción del usuario cuando hace submit de un formulario, o mejor dicho cuando hace una petición HTML POST al Portlet

➔ **render**: Ejecuta la vista VIEW, EDIT o HELP, según sea la solicitud del usuario.

➔ **doView**: Muestra la plantilla JSP correspondiente a la petición del usuario: Formulario de búsqueda, Formulario de categorías, etc...

➔ **doEdit**: Muestra la pantalla para la configuración del Portlet. En este proyecto no hay ningún Portlet que implemente la vista de edición.

➔ **doHelp**: Muestra la pantalla de ayuda del Portlet. En este proyecto no hay ningún Portlet que implemente la vista de ayuda.

➔ **destroy**: Método que se ejecuta cuando finaliza la ejecución del Portlet, liberando y cerrando los recursos usados en la ejecución del Portlet.

Los Portlets que heredan del dispatcher, lo único que hacen es llamar al método *init()* del padre. Es posible utilizar directamente el dispatcher, pero definiendo una clase para cada Portlet conseguimos que, en futuro, podamos extender las capacidades del dispatcher según las necesidades del Portlet.

4.4 Struts

Los Portlets desarrollado son Struts Portlet, tal y como se define en el punto anterior. Cuando se configura el dispatcher, se definen los namespaces y actions de Struts por defecto para cada vista del Portlet.

Para implementar las funcionalidades definidas para cada Portlet, se han definido una serie de Actions de Struts, cada uno de los cuales define un conjunto de operaciones relacionadas con la acción a realizar. Todos estos Actions son gestionados a través de Spring, siguiendo con el concepto de inyección de dependencia, de tal forma que se le puedan asignar los servicios de acceso a datos necesarios para el correcto funcionamiento de la aplicación.

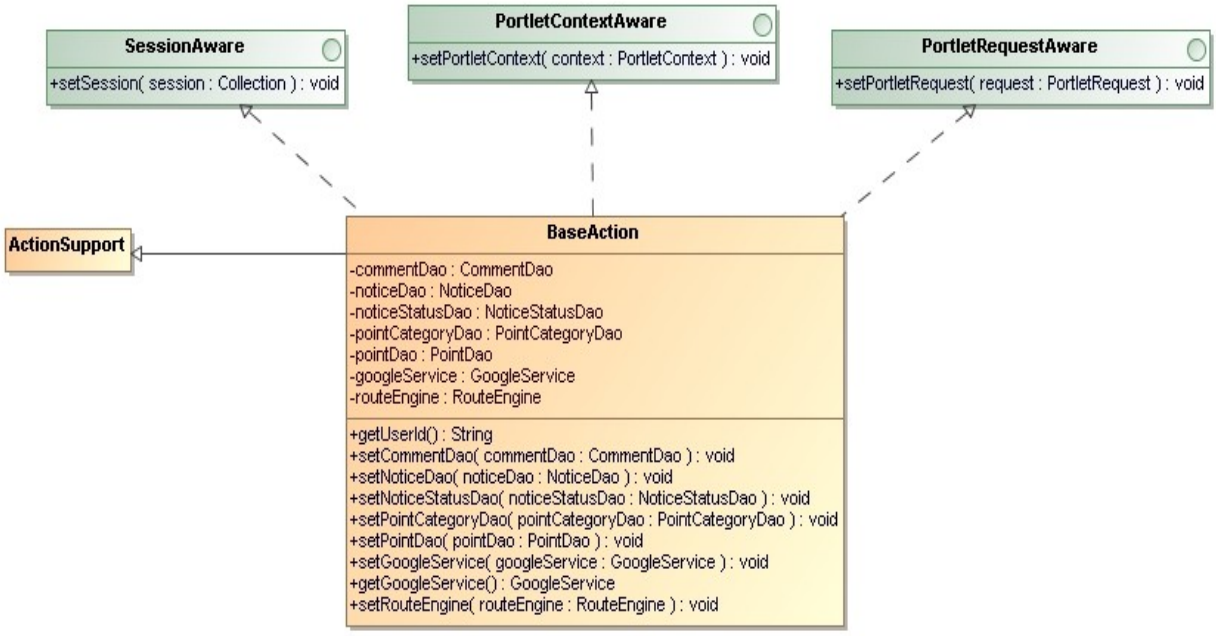

Diagrama 5: Acciones de Struts

Se ha definido una clase abstracta de la que heredan el resto de las acciones definidas para cada uno de los Portlets, tal y como se verá más adelante.

➔ ***BaseAction****:* Define una acción con las operaciones necesarias por el resto de aplicaciones.

- **getUserId**: Devuelve el identificador del usuario del portal que está ejecutando el Portlet.
- **setCommentDao**: Método utilizado para inyectar el servicio de acceso a datos para los comentarios
- **setNoticeDao**: Método utilizado para inyectar el servicio de acceso a datos para los avisos
- **setNoticeStatusDao**: Método utilizado para inyectar el servicio de acceso a datos para los estados de los avisos
- **setPointCategoryDao**: Método utilizado para inyectar el servicio de acceso a datos para las categorías de los puntos de interés
- **setPointDao**: Método utilizado para inyectar el servicio de acceso a datos para los puntos de interés
- **setGoogleService**: Método utilizado para inyectar el servicio de acceso a la API de Google
- **getGoogleService**: Devuelve el servicio de acceso a la API de Google
- **setRouteEngine**: Método utilizado para inyectar el motor de cálculo de rutas

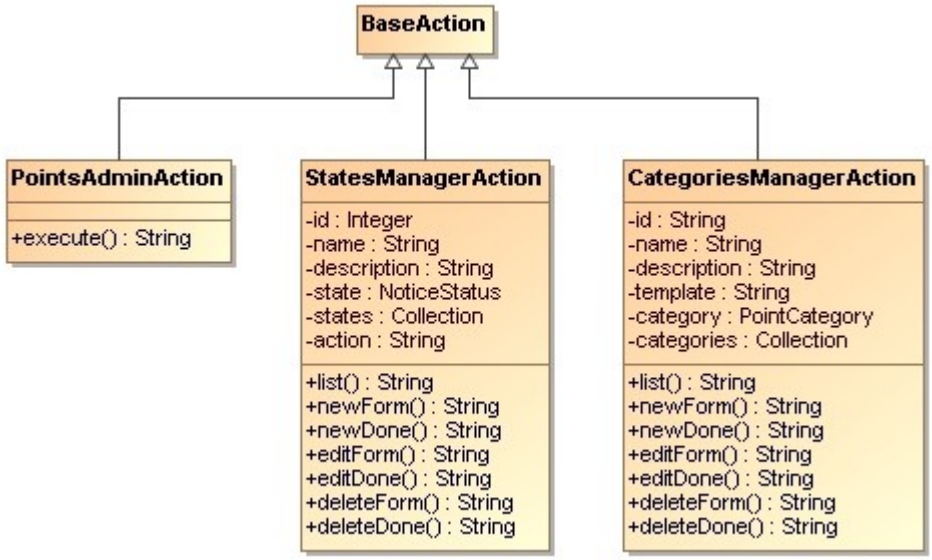

Diagrama 6: Acciones implicadas en el Portlet de Administración

➔ ***PointsAdminAction***: Define las operaciones por defecto del Portlet de Administración.

- **execute**: Devuelve la referencia a la pantalla inicial del Portlet de Administración.

➔ ***StatesManagerAction***: Define las operaciones para la gestión de estados.

- **list**: Devuelve la referencia a la pantalla de listado de estados
- **newForm**: Devuelve la referencia a la pantalla de alta de estado
- **newDone**: Guarda el nuevo estado y devuelve la referencia a la pantalla de información al usuario
- **editForm**: Carga el estado que se va a modificar y devuelve la referencia a la pantalla de edición de estado
- **editDone**: Guarda el estado modificado y devuelve la referencia a la pantalla de información al usuario
- **deleteForm**: Carga el estado que se va a eliminar y devuelve la referencia a la pantalla de eliminación de estado
- **deleteDone**: Elimina el estado y devuelve la referencia a la pantalla de información al usuario

➔ ***CategoriesManagerAction***: Define las operaciones para la gestión de categorías.

- **list**: Devuelve la referencia a la pantalla de listado de categorías
- **newForm**: Devuelve la referencia a la pantalla de alta de categoría
- **newDone**: Guarda la nueva categoría y devuelve la referencia a la pantalla de información al usuario

- **editForm**: Carga la categoría que se va a modificar y devuelve la referencia a la pantalla de edición de categoría
- **editDone**: Guarda la categoría modificada y devuelve la referencia a la pantalla de información al usuario
- **deleteForm**: Carga la categoría que se va a eliminar y devuelve la referencia a la pantalla de eliminación de categoría
- **deleteDone**: Elimina la categoría y devuelve la referencia a la pantalla de información al usuario

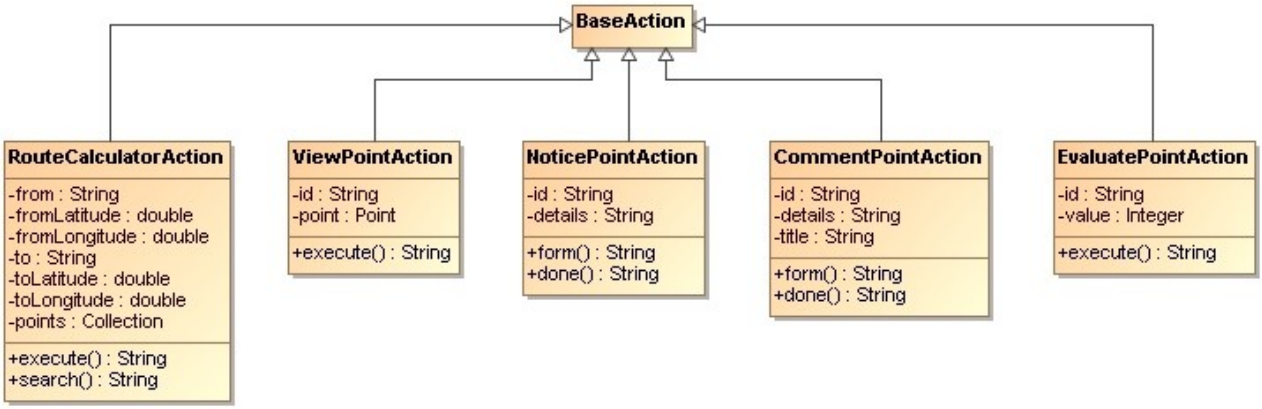

Diagrama 7: Acciones implicadas en el Portlet de Cálculo de Rutas

➔ ***RouteCalculatorAction****:* Define las operaciones por defecto del Portlet Calculadora de Rutas

- **execute**: Devuelve la referencia a la pantalla de cálculo de rutas
- **search**: Calcula una ruta con los parámetros definidos por el usuario y devuelve la referencia a la pantalla de resultados

➔ ***ViewPointAction***: Define las operaciones para visualizar puntos de interés

- **execute**: Carga el punto de interés a visualizar y devuelve la referencia a la página de visualización de punto de interés.

➔ ***NoticePointAction****:* Define las operaciones para notificar avisos en los puntos de interés

- **form**: Devuelve la referencia a la página del formulario de aviso
- **done**: Crea el aviso y devuelve la referencia a la página de información al usuario

➔ ***CommentPointAction***: Define las operaciones para comentar puntos de interés

- **form**: Devuelve la referencia la página del formulario de comentario
- **done**: Guarda el formulario y devuelve la referencia a la página de información al usuario

➔ ***EvaluatePointAction****:* Define las operaciones para evaluar puntos de interés

- **execute**: Evalua el punto de interés y devuelve la referencia a la página de información al usuario

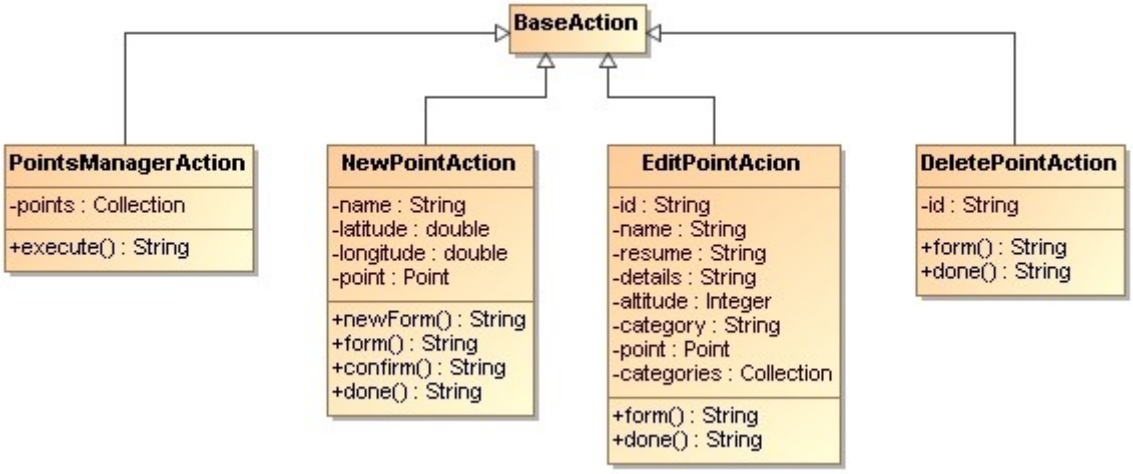

Diagrama 8: Acciones implicadas en el Portlet de Gestión de puntos

➔ ***PoinsManagerAction****:* Define las operaciones por defecto del Portlet de Gestión

- **execute**: Devuelve la referencia a la pantalla inicial del Portlet de Gestión

➔ ***NewPointAction****:* Define las operaciones para el alta de un punto de interés

- **form**: Devuelve la referencia a la pantalla de formulario de alta de un punto de interés
- **confirm**: Crea el nuevo punto en memoria y devuelve la referencia a la pantalla de confirmación de nuevo punto de interés
- **done**: Guarda el nuevo punto de interés y devuelve la referencia a la pantalla de información al usuario

➔ ***EditPointAction***: Define las operaciones para la edición de un punto de interés

- **form**: Carga el punto que se va a modificar y devuelve la referencia a la pantalla de edición de punto de interés
- done: Guarda el punto de interés modificado y devuelve la referencia a la pantalla de información al usuario

➔ ***DeletePointAction***: Define las operaciones para la eliminación de un punto de interés

- **form**: Carga el punto que se va a eliminar y devuelve la referencia a la pantalla de eliminación de punto de interés
- **done**: Elimina el punto de interés y devuelve la referencia a la pantalla de información al usuario

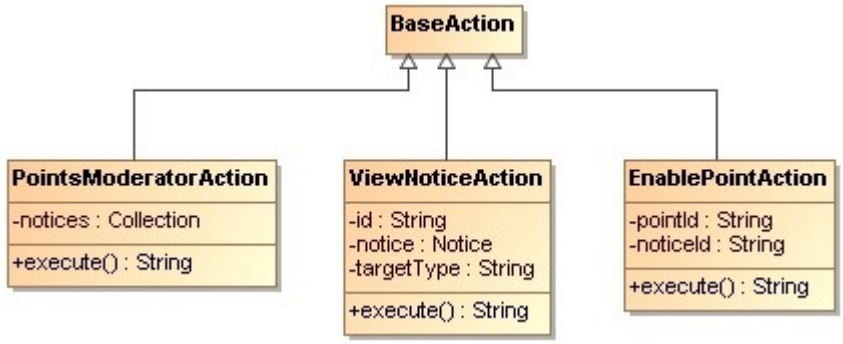

Diagrama 9: Acciones implicadas en el Portlet de Moderación

➔ ***PointsModeratorAction***: Define las operaciones por defecto del Portlet de Moderación

- **execute**: Devuelve la referencia a la pantalla inicial del Portlet de Moderación

➔ ***ViewNoticeAction***: Define las operaciones de visualización de avisos

- **execute**: Carga el aviso y devuelve la referencia a la pantalla de visualización de aviso

➔ ***EnablePointAction****:* Define las operaciones para habilitar puntos de interés

- **execute**: Carga el punto de interés, se habilita, se guarda y se devuelve la referencia a la pantalla de información al usuario.

4.5 Route Engine

Una de las características del proyecto es el cálculo de rutas turísticas. Para ello, además del propio motor *RouteEngine*, se han definido dos tipos de datos necesarios para realizar los cálculos.

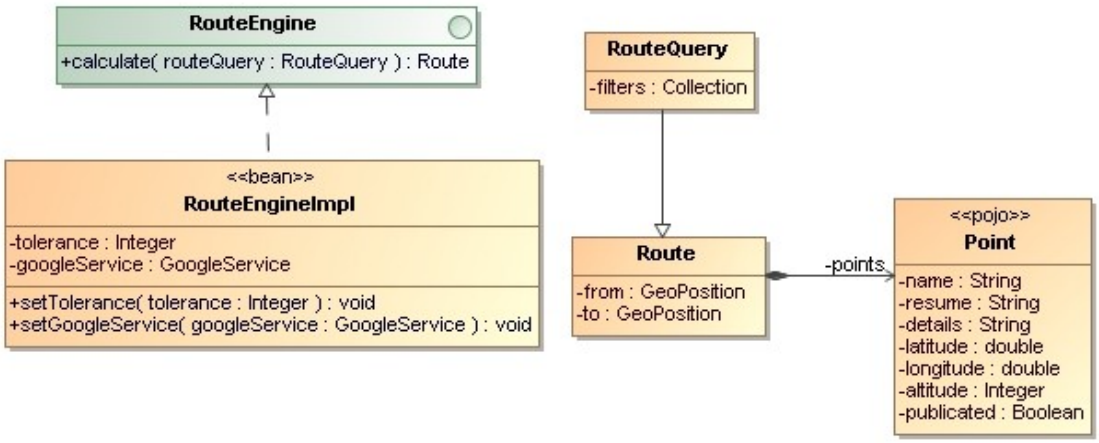

Diagrama 10: Clases del motor de cálculo de rutas

➔ ***Route***: Define los puntos de inicio y fin de la ruta además de un conjunto de puntos de interés que serán usados para almacenar los resultados obtenidos en el cálculo

➔ ***RouteQuery***: Hereda las propiedades de la clase *Route* y además define una colección de filtros que serán utilizados para realizar el cálculo de la ruta

Conociendo los tipos de datos necesarios para realizar el cálculo de ruta, se define por un lado la interfaz que tiene que implementar un motor de búsqueda y por otro lado se define una clase que implementa la interfaz para definir un motor de búsqueda básico:

➔ ***RouteEngine***: Interfaz que define el objeto motor de calculo de rutas

- **calculate**: Recibe un objeto *RouteQuery* con los parámetros de la búsqueda, realiza el cálculo correspondiente y devuelve un objeto *Route* con el resultado del cálculo de la ruta

➔ ***RouteEngineImpl***: Clase que implementa la interfaz *RouteEngine* para definir un motor de cálculo de rutas básico

- **setTolerance**: Define la tolerancía, a modo de desviación, de los puntos sobre la ruta resultante
- **setGoogleService**: Método para inyectar el servicio de acceso a la API de Google

4.6 Google Service

Este componentes es muy básico para los objetivos de este proyecto, pero se ha creado como una entidad con la intención de que pueda ser extendido con multitud de herramientas.

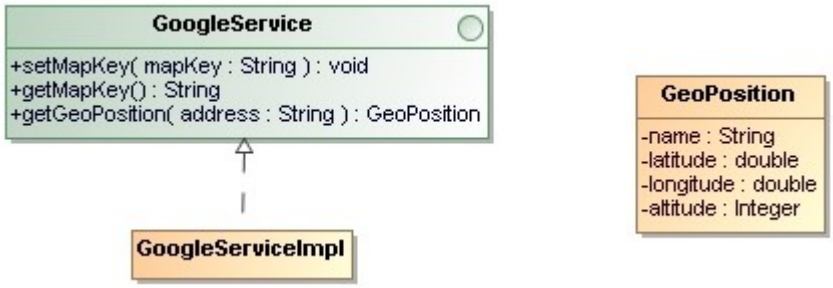

Diagrama 11: Clases del servicio de acceso a la API de Google

➔ ***GoogleService***: Define los servicios de acceso a la API de Google

- **setMapKey**: Establece la key Javascript para la ejecución de Google Maps
- **getMapKey**: Devuelve la key Javascript de Google Maps
- **getGeoPosition**: Devuelve la posición geográfica de una dirección postal

El servicio es lanzado desde Spring para ser inyectado en los componentes que así lo requieran, como puede ser la implementación básica del motor de rutas.

4.7 Diagramas de secuencia

Para finalizar la sección, y como ya conocemos las clases del sistema, veremos las secuencias más interesantes que se tienen que resolver.

4.7.1 Calcular una ruta

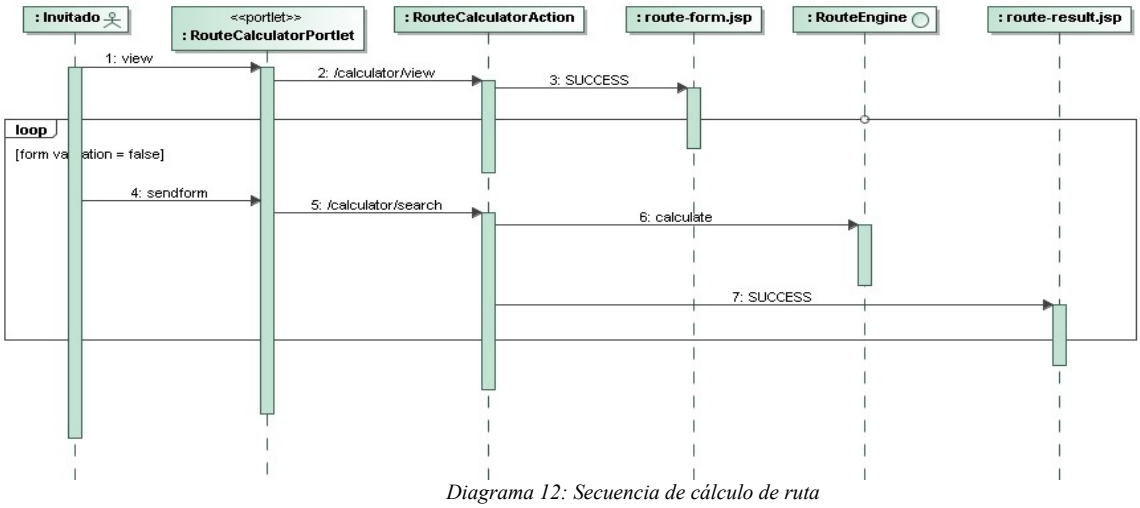

Diagrama 12: Secuencia de cálculo de ruta

El proceso de calculo de una ruta sigue la siguiente secuencia:

- ➔ En primer lugar, el usuario visualiza el Portlet de cálculo de rutas
- ➔ El Portlet ejecuta el Action /calculator/view definido por defecto, que devuelve la página con el formulario
- ➔ El usuario envía el formulario al Portlet
- ➔ Si el formulario se valida correctamente, se ejecuta la acción /calculator/search
- ➔ La acción /calculator/search utiliza el servicio *routeEngine* para realizar el cálculo y obtener los resultados
- ➔ Finalmente, se muestra al usuario la pantalla con los resultados

4.7.2 Visualización de un punto de interés

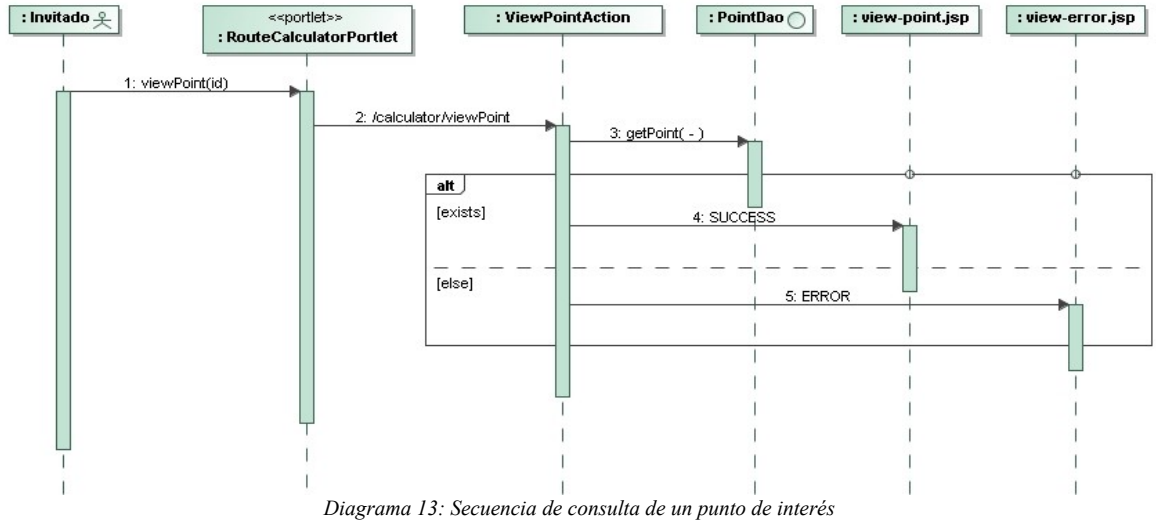

Diagrama 13: Secuencia de consulta de un punto de interés

La visualización de un punto de interés se divide en las siguientes operaciones:

- ➔ El usuario llama al Portlet de cálculo de rutas pasándole el identificador del punto que se quiere visualizar
- ➔ El Portlet ejecuta la acción /calculator/viewPoint que define la acción *ViewPointAction*
- ➔ La acción carga el punto usando el DAO de acceso a los puntos de interés
- ➔ Si el punto es encontrado, se muestra al usuario la pantalla de visualización de un punto de interés
- ➔ Si el punto no es válido, se muestra al usuario una pantalla informando del error

4.7.3 Crear un punto de interés

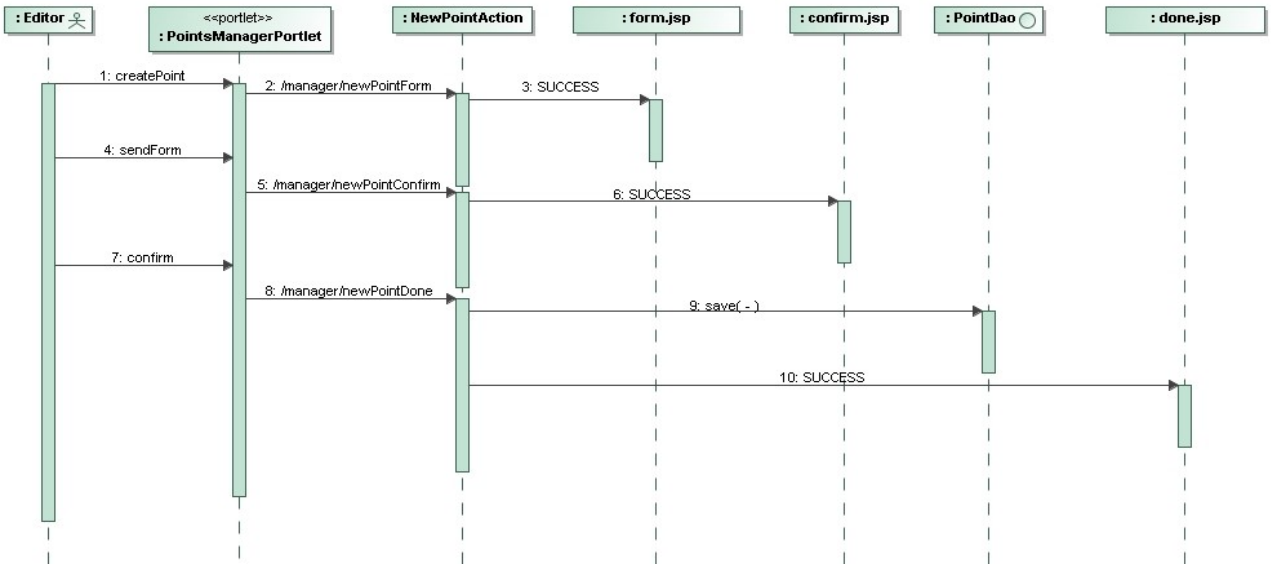

Diagrama 14: Secuencia de creación de un punto de interés

Para crear un punto de interés, se realizan las siguientes operaciones:

- ➔ El usuario Editor, llama al Portlet de gestión de puntos invocando la operación de nuevo punto de interés
- ➔ El Portlet llama a la acción /manager/newPointForm correspondiente a la operación de crear un nuevo punto que mostrará al usuario la pantalla con el formulario de alta
- ➔ El usuario envía el formulario con los datos y el Portlet llama a la acción /manager/newPointConfirm, que mostrará al usuario la pantalla de confirmación de alta de nuevo punto
- ➔ El usuario acepta y el Portlet llama a la acción /manager/newPointDone
- ➔ La última acción, registrará el nuevo punto a través del DAO de acceso a puntos de interés y mostrará al usuario la pantalla con información sobre la operación

4.7.4 Consulta de avisos

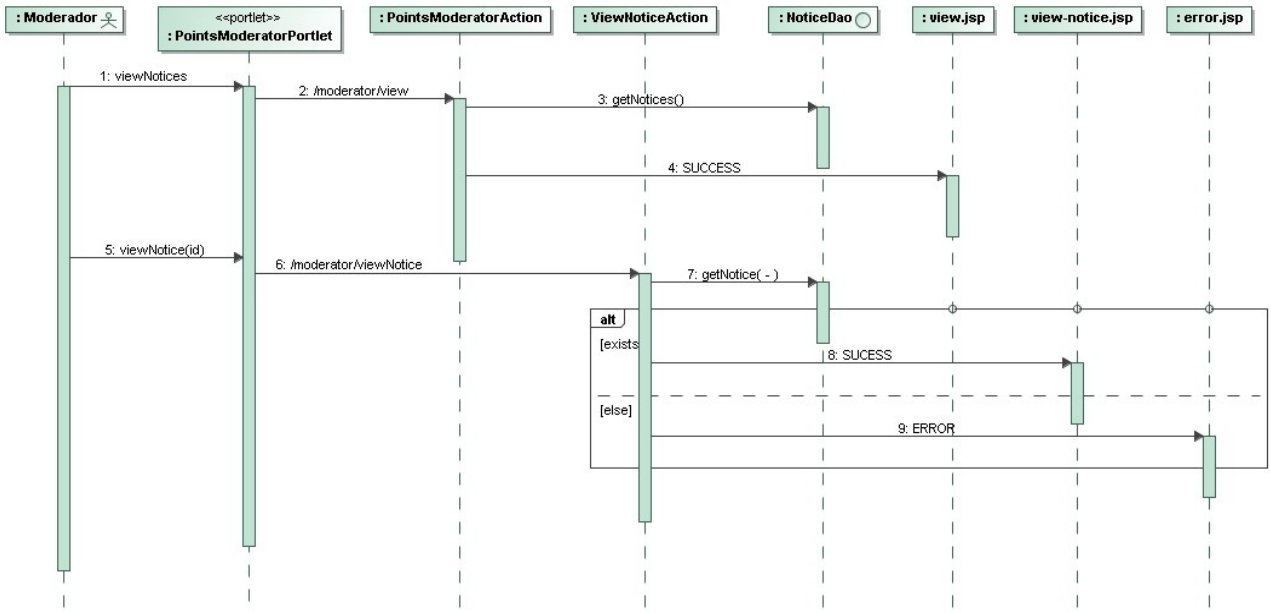

Diagrama 15: Secuencia de consulta de avisos

El proceso de consulta de avisos solo lo puede realizar el usuario Moderador. Las operaciones que se realizan son:

- ➔ El usuario moderador ejecuta la vista por defecto del Portlet de moderación, que es la acción /moderator/view
- ➔ La acción carga todos los avisos a través del DAO de acceso a avisos y muestra la pantalla con el listado de avisos al usuario
- ➔ El usuario solicita visualizar un determinado aviso especificando su identificador y el Portlet ejecuta la acción /moderator/viewNotice
- ➔ La acción, carga el aviso utilizando el DAO de acceso a avisos
- ➔ Si el aviso existe, se muestra al usuario la pantalla de visualización de aviso
- ➔ Si no existe, se muestra al usuario una pantalla informando del error

5. IMPLEMENTACIÓN

5.1 Herramientas de desarrollo

Para el desarrollo del proyecto se han utilizado, además de Java JEE 5, las siguientes herramientas:

- **Eclipse 3.4 Ganymede**: IDE por excelencia para desarrollo de proyectos Java en entornos corporativos. Incluye multitud de plugins para el desarrollo de aplicaciones Web Java JEE.
- **Apache Tomcat 6**: Contenedor de Servlets que cumple con los requerimientos mínimos para ejecutar la aplicación.
- **MySQL Server 5.0**: Servidor de bases de datos OpenSource.
- **Apache Ant 1.7.0**: Herramienta para realizar tareas reiterativas. Se utiliza a través de un lenguaje de scripting escrito en XML y se ha utilizado para crear las tareas de compilación, distribución, despliegue y limpieza del proyecto.
- **Subversion**: Servidor de control de versiones utilizado para guardar los cambios realizados durante el desarrollo del proyecto

Además de las herramienta utilizadas en el desarrollo del proyecto, se han utilizado las siguientes herramientas para el análisis y diseño:

- **OpenOffice.org 3**: Suite Ofimática OpenSource utilizada para la elaboración de toda la documentación entregada, incluida la presente memoria, además de las plantillas de presentación del proyecto.
- **MagicDraw UML 15.1**: Herramienta de modelado UML usada para elaborar los diagramas presentados en ésta memoria
- **Microsoft Project 2003**: Herramienta de planificación de proyectos

5.2 Estructura del proyecto Eclipse

El proyecto se estructura en los siguientes directorios principales:

- **dist**: Directorio donde se genera la distribución una vez ejecutada la tarea Ant.
- **doc**: Directorio dedicado a la documentación Javadoc de toda la API del proyecto
- **lib**: Librerías 3rd Party
- **src**: Código del proyecto

El directorio de fuentes se divide en tres subdirectorio:

- **java**: Código fuente de todas las clases Java del proyecto
- **resources**: Directorio con todos los ficheros de configuración
- **routecalculator**: Proyecto Web

Para generar la distribución y realizar las tareas de compilación, se utiliza Apache Ant. Se han desarrollado tres tareas principales:

- **clean**: Elimina los ficheros compilados y la distribución previamente generada
- **deploy**: Despliega la distribución en un servidor Apache Tomcat existente
- **dist**: Genera la distribución de la aplicación

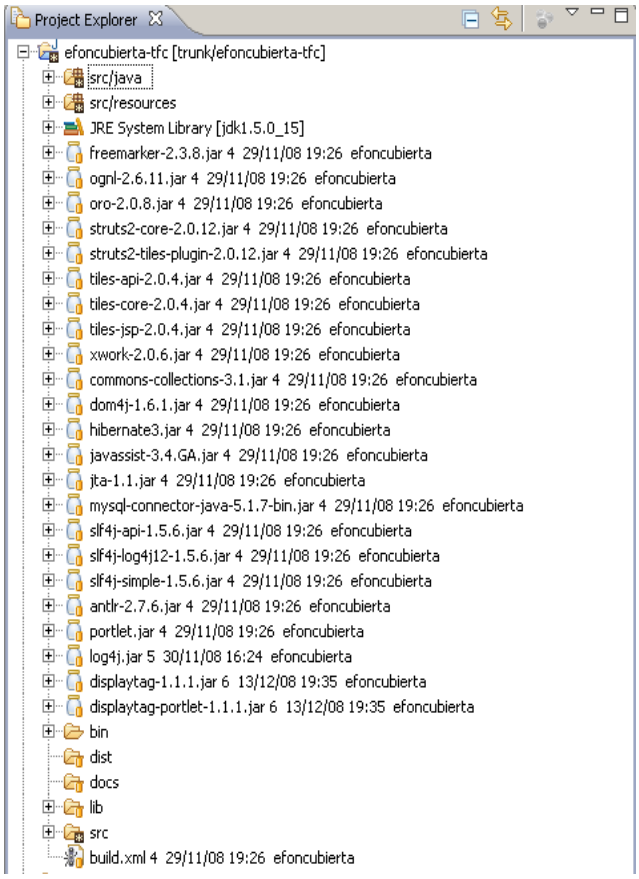

Ilustración 6: Estructura de proyecto Eclipse

5.3 Librerías

A continuación se definen algunas de las librerías más importantes para el correcto funcionamiento de la aplicación.

Desde el proyecto Struts, son necesarias:

- **freemarker-2.3.8.jar**: Soporte para plantillas Freemarker
- **ognl-2.6.11.jar**: Soporte para Object Graph Navigation Language, usado por Struts para la navegación por las propiedades de los objetos
- **oro-2.0.8.jar**: Conjunto de herramientas para el procesamiento de expresiones regulares
- **struts2-core-2.0.12.jar**: Librería principal de Struts 2
- **xwork-2.0.6.jar**: Librería heredada del proyecto WebWork que incluye el core de dicho framework
- **struts2-spring-plugin-2.0.12.jar**: Soporte Spring en Struts para poder definir las acciones como beans de Spring

Desde el proyecto Spring, son necesarias las librerías:

- **spring.jar**: Librería principal del framework Spring

- **spring-jdbc.jar**: Soporte JDBC para Spring
- **spring-orm.jar**: Soporte de frameworks de persistencia para Spring.

5.4 Configuración de la Aplicación

5.4.1 Aplicación Web

La configuración de esta aplicación Web es muy simple.

web.xml

```
<web-app>
    <display-name>routecalculator-plugin</display-name>
    <description>Route Calculator</description>

    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:spring/*-context.xml</param-value>
    </context-param>

    <filter>
        <filter-name>struts2</filter-name>
        <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-class>
    </filter>

    <filter-mapping>
        <filter-name>struts2</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>
</web-app>
```

Como se puede ver, la configuración de la aplicación Web consta de dos partes. Por un lado, se define *FilterDispatcher*, un filtro Web de Struts 2, donde se mapean todas las peticiones a la aplicación. Por otro lado, se define *ContextLoaderListener*, un listener de Spring, el cual inicializa el contexto de la aplicación Web en Spring. Además, se define para Spring el parámetro *contextConfigLocation* para que Spring cargue su factoría con los beans definidos en los ficheros con extensión *-context.xml*.

5.4.2 Portlets

La configuración de los componentes Webs también es muy sencilla. Consiste, principalmente, en la definición de los componentes en un fichero de descripción de Portlets que se encuentra en el directorio *src/routecalculator/WEB-INF/portlet.xml*.

portlet.xml

```
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1.0"
             xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd">
    <portlet>
        <description>Points Administration</description>
        <portlet-name>EF_POINTS_ADMIN</portlet-name>
        <portlet-class>uoc.edu.efoncubierta.portlet.PointsAdminPortlet</portlet-class>
        <init-param>
            <name>viewNamespace</name>
            <value>/admin</value>
```

```
        </init-param>
        <init-param>
            <name>defaultViewAction</name>
            <value>view</value>
        </init-param>
        <supports>
            <mime-type>text/html</mime-type>
            <portlet-mode>VIEW</portlet-mode>
        </supports>
        <portlet-info>
            <title>Points Administration</title>
            <short-title>Points Administration</short-title>
        </portlet-info>
    </portlet>
    <portlet>
        <description>Points Manager</description>
        <portlet-name>EF_POINTS_MANAGER</portlet-name>
        <portlet-class>uoc.edu.efoncubierta.portlet.PointsManagerPortlet</portlet-class>
        <init-param>
            <name>viewNamespace</name>
            <value>/manager</value>
        </init-param>
        <init-param>
            <name>defaultViewAction</name>
            <value>view</value>
        </init-param>
        <supports>
            <mime-type>text/html</mime-type>
            <portlet-mode>VIEW</portlet-mode>
        </supports>
        <portlet-info>
            <title>Points Manager</title>
            <short-title>Points Manager</short-title>
        </portlet-info>
    </portlet>
    <portlet>
        <description>Points Moderator</description>
        <portlet-name>EF_POINTS_MODERATOR</portlet-name>
        <portlet-class>uoc.edu.efoncubierta.portlet.PointsModeratorPortlet</portlet-class>
        <init-param>
            <name>viewNamespace</name>
            <value>/moderator</value>
        </init-param>
        <init-param>
            <name>defaultViewAction</name>
            <value>view</value>
        </init-param>
        <supports>
            <mime-type>text/html</mime-type>
            <portlet-mode>VIEW</portlet-mode>
        </supports>
        <portlet-info>
            <title>Points Moderator</title>
            <short-title>Points Moderator</short-title>
        </portlet-info>
    </portlet>
    <portlet>
        <description>Route Calculator</description>
        <portlet-name>EF_ROUTE_CALCULATOR</portlet-name>
        <portlet-class>uoc.edu.efoncubierta.portlet.RouteCalculatorPortlet</portlet-class>
        <init-param>
            <name>viewNamespace</name>
            <value>/calculator</value>
        </init-param>
        <init-param>
            <name>defaultViewAction</name>
            <value>view</value>
        </init-param>
        <supports>
            <mime-type>text/html</mime-type>
            <portlet-mode>VIEW</portlet-mode>
        </supports>
        <portlet-info>
            <title>Route Calculator</title>
```

```
            <short-title>Route Calculator</short-title>
        </portlet-info>
    </portlet>
    <user-attribute>
        <description>User Nick Name</description>
        <name>user.name.nickName</name>
    </user-attribute>
</portlet-app>
```

En este fichero se definen los cuatro Portlets. Para cada uno de ellos, se define su clase y los parámetros de incio *viewNamespace* y *defaultViewAction*. Estos parámetros indican el namespace de Struts donde se encuentra el Action por defecto para la vista VIEW. Por ejemplo, cuando se ejecuta la vista VIEW del Portlet *Route Calculator*, realmente se está llamando a la URL ***/calculator/view***, que, como veremos más adelante, ejecutará el Action correspondiente a dicho mapeo.

5.4.3 Struts

La configuración de Struts 2 se define en los ficheros *struts-*.xml* que se encuentran en el directorio *resources*. Explicaremos, para empezar, la configuración del fichero *struts.xml*:

struts.xml

```
<struts>
    <constant name="struts.objectFactory"
              value="org.apache.struts2.spring.StrutsSpringObjectFactory" />

    <package name="tfc-default" extends="struts-portlet-default" abstract="true">
        <global-results>
            <result name="exception">/jsp/error.jsp</result>
        </global-results>

        <global-exception-mappings>
            <exception-mapping exception="java.lang.Exception" result="exception" />
        </global-exception-mappings>
    </package>

    <include file="struts-portlet-default.xml" />
    <include file="struts-portlet-admin.xml" />
    <include file="struts-portlet-calculator.xml" />
    <include file="struts-portlet-manager.xml" />
    <include file="struts-portlet-moderator.xml" />
</struts>
```

En primer lugar, se define la constante *struts.objectFactory* que define la factoría de objetos que utilizará Struts. Lo normal es no definir esta constante, siendo Struts el que instancie cada uno de los Actions, pero como los Actions de Struts de definen como beans de Spring, entonces a través de esta constante le decimos que los Actions se encuentran en la factoría de Spring.

En Struts 2 se introduce el concepto de paquete, cuyo objetivo es definir un conjunto de acciones, además de una serie de configuraciones. Estos paquetes pueden ser abstractos, de forma que tenga una configuración pero cuyas acciones no podrán ser llamadas. De esta forma, se pueden crear paquetes globales del cual hereden el resto de paquetes de la aplicación, teniendo así la configuración global definida en un único punto. Es por ello que se crea el paquete *tfc-default* que hereda a su vez de *struts-portlet-default*, cuyo paquete define los interceptores necesarios para manejar el contexto de ejecución de los Portlets.

Por último, el fichero incluye otros ficheros xml, uno por cada Portlet, de forma que la configuración de Struts queda modularizada en varios ficheros, mejorando su mantenimiento.

- **struts-portlet-default.xml**: Fichero que define el paquete *struts-portlet-default*, que incluye la configuración necesaria de los interceptores para manejar los contextos de los Portlets
- **struts-portlet-admin.xml**: Fichero que define las acciones del Portlet de administración.
- **struts-portlet-calculator.xml**: Fichero que define las acciones del Portlet de cálculo de rutas
- **struts-portlet-manager.xml**: Fichero que define las acciones de Portlet de gestión de puntos
- **struts-portlet-moderator.xml**: Fichero que define las acciones del Portlet de moderación

A continuación veremos una parte del fichero *struts-portlet-calculator.xml*. El resto de ficheros siguen la misma estructura.

struts-portlet-calculator.xml

```
<struts>
    <package name="calculator" extends="tfc-default" namespace="/calculator">
        <action name="view" class="routeCalculatorAction">
            <result>/jsp/calculator/view.jsp</result>
        </action>

        <action name="search" class="routeCalculatorAction" method="search">
            <result>/jsp/calculator/result.jsp</result>
            <result name="input">/jsp/calculator/view.jsp</result>
        </action>
    </package>

    <!-- Resto de acciones -->
</struts>
```

El fichero define el paquete *calculator* que hereda de *tfc-default* y cuyo namespace es */calculator*. El paquete define todos los actions del Portlet Calculadora de Rutas, aunque en esta muestra solo vemos dos. En primer lugar, se define el action *view* cuya clase es *routeCalculatorAction*. Como vimos antes, el Portlet Calculadora llamaba por defecto a la url **/calculator/view** que es precisamente el primer action definido en este fichero.

La clase del Action no es una clase como tal, ya que al utilizar el plugin de Spring, en lugar de definir una clase en el atributo class, se da una referencia a un bean de Spring definido en el fichero *src/resources/spring/struts-actions-context.xml*. En este fichero, se define el bean *baseAction* cuyas propiedades son todos los DAOs, Google Service y el Route Engine. De este bean tienen que heredar todos los Actions declarados como beans de Spring.

struts-actions-context.xml

```
<beans default-autowire="autodetect">
    <bean id="baseAction" abstract="true" class="uoc.edu.efoncubierta.struts.BaseAction">
        <property name="commentDao">
            <ref bean="commentDao"/>
        </property>
        <property name="noticeDao">
            <ref bean="noticeDao"/>
        </property>
        <property name="noticeStatusDao">
```

```
            <ref bean="noticeStatusDao"/>
        </property>
        <property name="pointCategoryDao">
            <ref bean="pointCategoryDao"/>
        </property>
        <property name="pointDao">
            <ref bean="pointDao"/>
        </property>
        <property name="googleService">
            <ref bean="googleService"/>
        </property>
        <property name="routeEngine">
            <ref bean="routeEngine"/>
        </property>
    </bean>

    <!-- Resto de Acciones de Struts -->

    <bean id="routeCalculatorAction" parent="baseAction"
          class="uoc.edu.efoncubierta.struts.calculator.RouteCalculatorAction" singleton="false"/>

    <!-- Resto de Acciones de Struts -->
</beans>
```

Las acciones no se declaran como Singleton, ya que cada usuario tiene que ejecutar su propia instancia del Action para no compartir el contexto de ejecución.

5.4.4 Hibernate

La configuración de Hibernate se realiza completamente a través de Spring. Para ello, se ha creado un fichero *hibernate-context.xml* que define todos los servicios necesarios para la persistencia de los datos.

hibernate-context.xml

```
<beans>
    <bean name="hibernatePropertiesConfigurer"
          class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <property name="ignoreUnresolvablePlaceholders">
            <value>true</value>
        </property>

        <property name="locations">
            <value>classpath:spring/hibernate.properties</value>
        </property>
    </bean>

    <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName">
            <value>${db.driver}</value>
        </property>

        <property name="url">
            <value>${db.url}</value>
        </property>

        <property name="username">
            <value>${db.username}</value>
        </property>

        <property name="password">
            <value>${db.password}</value>
        </property>
    </bean>

    <bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
        <property name="dataSource">
            <ref local="dataSource" />
        </property>
```

```
        <property name="mappingResources">
            <list>
                <value>mappings/NoticeStatus.hbm.xml</value>
                <value>mappings/PointValoration.hbm.xml</value>
                <value>mappings/Node.hbm.xml</value>
            </list>
        </property>

        <property name="hibernateProperties">
            <props>
                <prop key="hibernate.dialect">
                    ${hibernate.dialect}
                </prop>
                <prop key="hibernate.hbm2ddl.auto">
                    ${hibernate.hbm2ddl.auto}
                </prop>
                <prop key="hibernate.show_sql">
                    ${hibernate.show_sql}
                </prop>
            </props>
        </property>
    </bean>

    <bean id="transactionManager"
          class="org.springframework.orm.hibernate3.HibernateTransactionManager">
        <property name="sessionFactory">
            <ref local="sessionFactory"/>
        </property>
    </bean>

    <bean id="commentDao" class="uoc.edu.efoncubierta.persistence.CommentDaoImpl">
        <property name="sessionFactory">
            <ref local="sessionFactory" />
        </property>
    </bean>

    <!-- Resto de DAOS -->
</beans>
```

En primer lugar, antes de definir cada servicio, se define un configurador de propiedades *hibernatePropertiesConfigurer*, que lee el fichero *hibernate.properties* donde se definen todos los parámetros que se le pasan a los servicios para su correcto funcionamiento.

hibernate.properties

```
# Datasource
db.driver = org.gjt.mm.mysql.Driver
db.url = jdbc:mysql://localhost/efoncubiertadb
db.username = efoncubierta
db.password = 123efoncubierta321

# Propiedades de hibernate
hibernate.dialect = org.hibernate.dialect.MySQL5InnoDBDialect
hibernate.show_sql = true
hibernate.hbm2ddl.auto = update
```

El fichero define por un lado los datos de conexión a la base de datos (parámetros db.*) y por otro lado los parámetros de configuración de Hibernate (parámetros hibernate.*).

Los beans definidos para Hibernate son:

- **dataSource**: Define un datasource con la conexión a la base de datos. Para instanciar el bean, se le pasan los datos de conexión a la base de datos definidos en el fichero de propiedades

- **sessionFactory**: Define una factoría con toda la configuración de Hibernate, a la que se le inyecta el dataSource con la conexión a la base de datos y los fichero de mappings correspondientes
- **transactionManager**: Define un servicio para el control de transacciones en Hibernate
- ***Dao**: Todos los servicios de acceso a datos son definidos en Spring para poder ser inyectados en aquellos componentes que necesitan acceder a los datos

Los ficheros de mapeos de Hibernate, pasados al servicio *sessionFactory*, definen las correspondencias entre las clases del dominio con las tablas de la base de datos.

- **NoticeStatus.hbm.xml**: Define los mapeos para los objetos *NoticeStatus*
- **PointValoration.hbm.xml**: Define los mapeos para los objetos *PointValoration*
- **Node.hbm.xml**: Define los mapeos para los objetos *Node, Point, PointCategory, Comment y Notice*

5.4.5 Google Service

El servicio de Google también se define como un bean de Spring en el fichero *google-context.xml*.

google-context.xml

```
<beans>
    <bean name="googlePropertiesConfigurer"
          class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <property name="ignoreUnresolvablePlaceholders">
            <value>true</value>
        </property>

        <property name="locations">
            <value>classpath:spring/google.properties</value>
        </property>
    </bean>

    <bean name="googleService" class="uoc.edu.efoncubierta.google.GoogleServiceImpl">
        <property name="mapKey">
            <value>${google.map.key}</value>
        </property>
    </bean>
</beans>
```

Por un lado, se incluye la configuración a través de *googlePropertiesConfigurer*. Por otro lado, se define el servicio *googleService*, pasándole por parámetro las propiedades definidas en el fichero *google.properties*, que para el alcance de este proyecto se basa en una única propiedad.

google.properties

```
google.map.key = ABQIAAAAzdkgVGPSCRk3j4S4E_f5LRTwM0brOpm-All5BF6PoaKBxRWWERQ0yYaTNEdh011vzeI539yJQUEbDQ
```

La propiedad *google.map.key* define la clave necesaria para ejecutar la API Javascript de Google Maps.

5.4.6 Route Engine

El motor de rutas también se configura a través de Spring. Al igual que con el servicio de Google, el motor de rutas se define en un fichero *calculator-context.xml*.

calculator-context.xml

```
<beans>
    <bean name="calculatorPropertiesConfigurer"
          class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <property name="ignoreUnresolvablePlaceholders">
            <value>true</value>
        </property>

        <property name="locations">
            <value>classpath:spring/calculator.properties</value>
        </property>
    </bean>

    <bean id="routeEngine" class="uoc.edu.efoncubierta.calculator.RouteEngineImpl">
        <property name="tolerance">
            <value type="int">${calculator.tolerance}</value>
        </property>
        <property name="googleService">
            <ref bean="googleService"/>
        </property>
    </bean>
</beans>
```

Por un lado, se define el conjunto de propiedades, definidas en el fichero *calculator.properties*, que se van a utilizar en el motor de rutas y por otro lado se define el motor con las propiedades necesarias. Para el caso del motor básico, solo se define la tolerancia permitida, en kilómetros, de los puntos resultantes sobre la ruta.

5.5 Proceso de instalación

Para facilitar el proceso de instalación, además de definir las tareas que se han de realizar para el despliegue en un entorno de producción existente, se ofrecen alternativas para realizar pruebas de la herramienta de una forma mucho más fácil. Si así lo desea, consulta directamente la sección “5.5.2 Instalar Bundle”.

5.5.1 Instalación normal

La instalación normal es la que se tiene que realizar cuando se va a desplegar la aplicación en un entorno de producción. Para ello es necesario comenzar generando la distribución de la aplicación y posteriormente instalar Liferay Portal y desplegar el proyecto sobre él.

Descargando dependencias

El proyecto se distribuye sin las librerías necesarias para su correcta ejecución, por lo que es necesario descargar cada una de ellas y posteriormente copiarlas en el directorio *src/routecalculator/WEB-INF/lib*, antes de hacer la distribución. Se ofrece la posibilidad de que el usuario descargue, si lo desea, todas las librerías necesarias de una sola vez desde la siguiente dirección http://www.frilan.net/tfc-libs.zip. En todo caso, las tareas a realizar para la instalación de las librerías son:

➔ Struts

- Descargamos desde http://apache.rediris.es/struts/library/struts-2.0.12-lib.zip
- Descomprimimos el paquete descargado
- Copiamos *freemarker-2.3.8.jar, ognl-2.6.11.jar, oro-2.0.8.jar, struts2-core-2.0.12.jar, xwork-2.0.6.jar y struts2-spring-plugin-2.0.12.jar* al directorio de librerías.

➔ Spring

- Descargamos desde http://www.springsource.com/download/community?project=Spring%20Framework
- Descomprimimos el paquete descargado
- Copiamos *spring.jar, spring-jdbc.jar y spring-orm.jar* al directorio de librerías

➔ Hibernate

- Descargamos desde http://www.hibernate.org/6.html la versión 3.3.1GA
- Descomprimimos el paquete descargado
- Copiamos *hibernate3.jar* al directorio de librerías
- Copiamos *antlr-2.7.6.jar, commons-collections-3.1.jar, dom4j-1.6.1.jar, javassist-3.4.GA.jar, jta-1.1.jar y slf4j-api-1.5.6.jar* desde el directorio *lib/required* al directorio de librerías.

➔ Slf4j

- Descargamos desde http://www.slf4j.org/download.html la versión 1.5.6
- Descomprimimos el paquete descargado
- Copiamos *slf4j-log4j12-1.5.6.jar y slf4j-simple-1.5.6.jar* al directorio de librerías

➔ MySQL Connector

- Descargamos desde http://dev.mysql.com/downloads/connector/j/5.0.html
- Descomprimimos el paquete descargado
- Copiamos *mysql-connector-java-5.1.7-bin.jar* al directorio de librerías

➔ Log4j

- Descargamos desde http://logging.apache.org/log4j/1.2/download.html la versión 1.2.15
- Descomprimimos el paquete descargado

- Copiamos *log4j.jar* al directorio de librerías

➔ DisplayTags

- Descargamos desde http://displaytag.sourceforge.net/11/download.html la versión 1.1.1
- Descomprimimos el paquete descargado
- Copiamos *displaytag-1.1.1.jar y displaytag-portlet-1.1.1.jar* al directorio de librerías

Generando la distribución

Con las librerías copiadas, el siguiente paso es editar el fichero *src/resources/spring/hibernate.properties* para definir los parámetro de conexión con la base de datos. Una vez configurado, se ejecuta la siguiente tarea Ant:

```
ant dist
```

Como resultado, obtendremos en el directorio *dist* la aplicación *routecalculator-plugin.war*

Instalando Liferay Portal

Para poder utilizar los portlets, necesitamos el contenedor:

➔ Descargar Liferay Portal desde http://downloads.sourceforge.net/lportal/liferay-portal-tomcat-5.5-5.1.2.zip . Este paquete ya incluye tomcat y está configurado y listo para ser usado.

➔ Descomprimir el paquete descargado

➔ Iniciar Tomcat ejecutando *tomcat/bin/startup.bat* (Si estamos en Linux, ejecutamos el script startup.sh)

Despliegue de los Portlets en Liferay

Con Liferay funcionando, copiar la aplicación *routecalculator-plugin.war* al directorio *liferay/deploy* del directorio personal del usuario.

```
En windows C:\Documents and Settings\efoncubierta\liferay\deploy
En linux /home/efoncubierta/liferay/deploy
```

El plugin será iniciado por Liferay automáticamente y dejará el sistema listo para ser utilizado.

5.5.2 Instalar Bundle

Esta instalación es aconsejable para hacer pruebas de funcionamiento sin necesidad de realizar el costoso proceso anterior. El paquete bundle ya incluye Tomcat y está configurado y listo para ser utilizado:

➔ Descargar desde http://www.frilan.net/tfc-bundle.zip

- ➔ Descomprimir el paquete descargado
- ➔ Iniciar tomcat ejecutando el script *tomcat/bin/startup.bat* (Si estamos en Linux, ejecutar el script startup.sh)

Las aplicaciones están configuradas para ser usadas con HSQL, de tal modo que no es necesaria ninguna configuración adicional de la base de datos.

5.6 Pruebas

Una vez lanzado el servidor, tal y como se explica en el punto anterior, se pueden comenzar a realizar pruebas de la aplicación. Para ello lanzamos la aplicación desde el navegador:

http://localhost:8080

Se iniciará Liferay y la primera tarea que hay tenemos que hacer es identificarnos en el sistema. Para autenticarse, utilizar el usuario *test@liferay.com* con password *test* y aceptar los términos y condiciones si es la primera vez que nos autenticamos.

La aplicación, lógicamente, no lanza los Portlets que hemos desarrollado. Para ello, en el menú superior tenemos el siguiente menú desplegable, conocido como *Dock*.

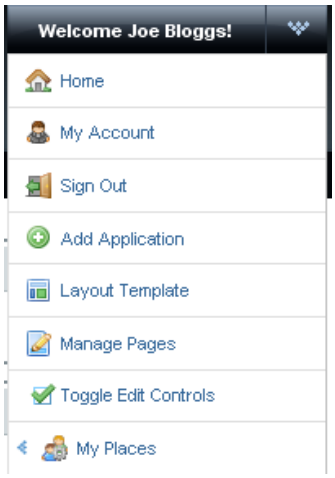

Ilustración 7: Menu Dock de Liferay

La que nos interesa en este momento, es la opción *Add Application*, que nos abrirá un menú a la izquierda de la pantalla con todos los Portlet que está gestionando Liferay. En dicho menú, buscamos la sección Efoncubierta → TFC, donde nos aparecerán los cuatro Portlet de este proyecto.

Pinchamos sobre el que queremos incluir y se cargará, a través de Ajax, en la página que estamos visualizando.

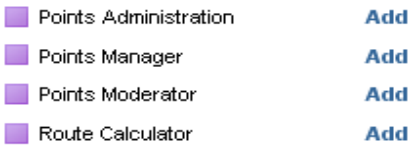

Ilustración 8: Menú con los Portlets del Proyecto

Una vez que tenemos el Portlet cargado en la página, si pinchamos en la barra del título y arrastramos, podremos posicionarlo en cualquier parte de la página.

5.6.1 Points Admin

La primera vez que accedemos no hay ningún dato creado, así que nuestra primera tarea será preparar el entorno.

Ilustración 9: Pantalla inicial del Portlet de Administración

La tarea del Administrador, para el alcance del proyecto, consiste en gestionar los estados de los avisos y las categorías de los puntos. Para comenzar, empezaremos gestionando los estados.

Ilustración 10: Pantalla de listado de estados vacío

Al comenzar no existe ningún estado, así que crearemos los siguientes:

- **open**: Estado para los avisos creados
- **closed**: Estado para los avisos cerrados
- **declined**: Estado para los avisos rechazados

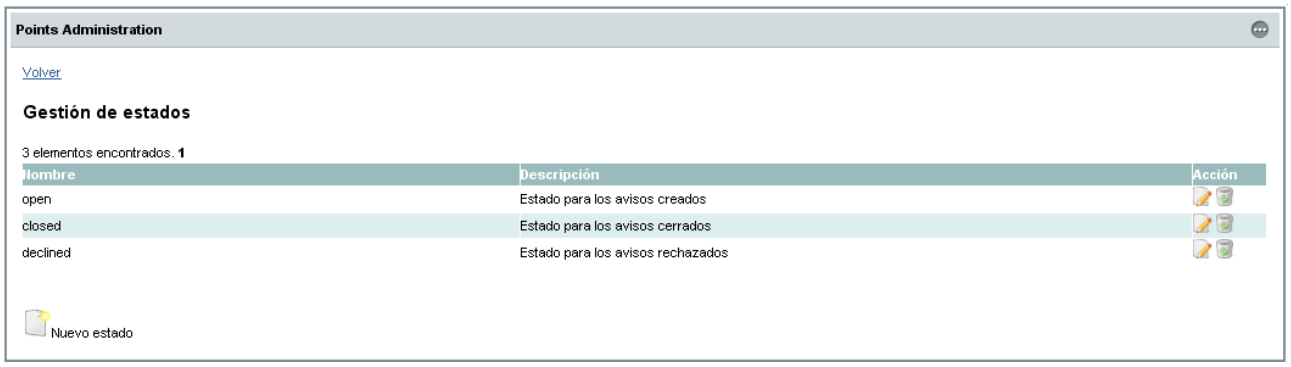

Ilustración 11: Pantalla de listado de estados

Con los estados creados, podemos eliminarlos o editarlos con las acciones que están a la derecha de la tabla. Pulsamos el enlace Volver, para ir a las opciones de Administración. Ahora crearemos un categoría:

- Nombre**: Castillo**
- Descripción: **Categoría para catalogar castillos**
- Template**: castle.ftl**

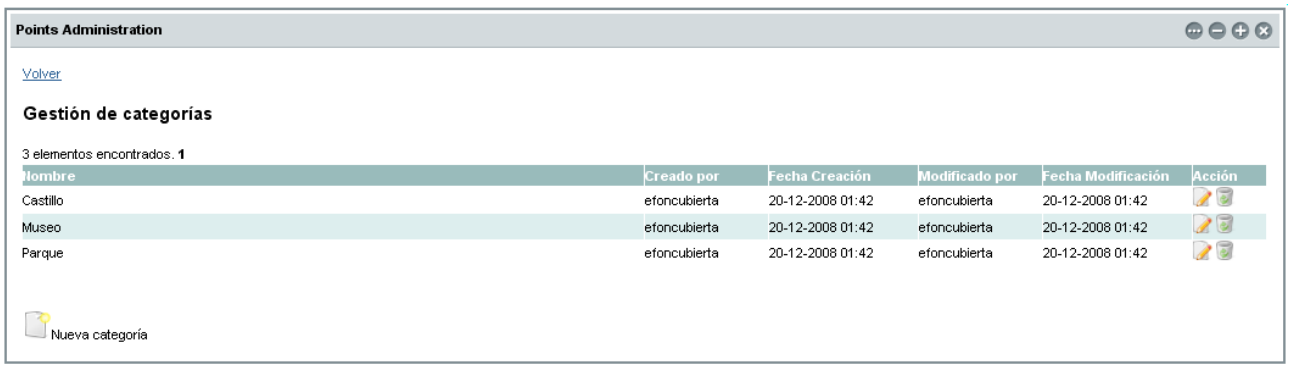

Ilustración 12: Pantalla de listado de categorías

Al crear una categoría, es necesario que la plantilla se encuentre en el directorio *templates* de la aplicación Web. En principio solo existen las plantillas *castle, museum y park*.

A partir de ahora, con la configuración previa del entorno realizada, ya se puede comenzar a trabajar con el resto de componentes.

5.6.2 Point Manager

La gestión de puntos se encarga de crear y editar los puntos del sistema. Es requisito indispensable que el usuario esté autenticado en el sistema para poder crear, o modificar, los puntos de interés.

Ilustración 13: Pantalla de listado de puntos de interés vacía

La primera vez no existirá ningún punto de interés creado. Pulsamos sobre "Nuevo punto" y creamos el punto en cuestión:

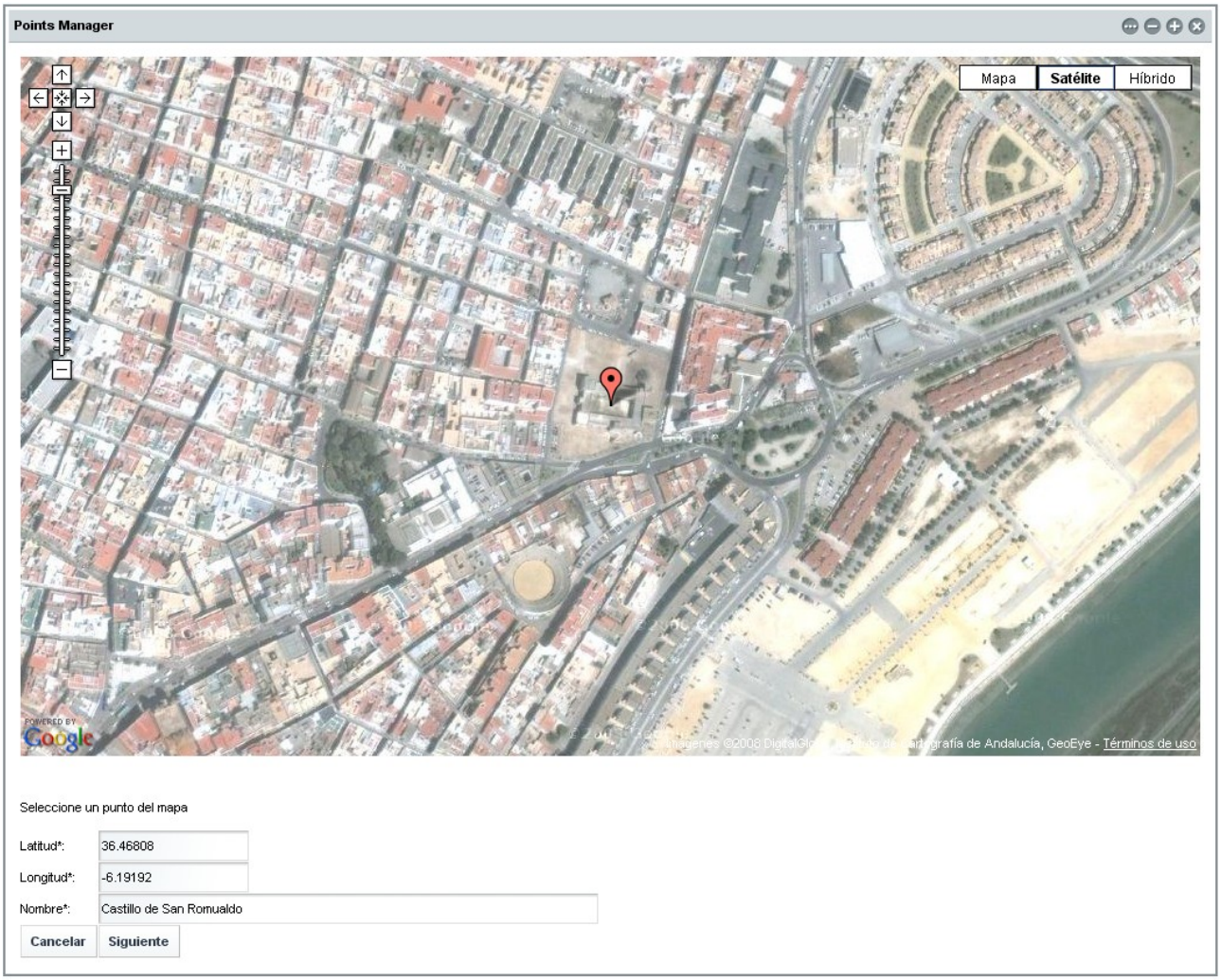

Ilustración 14: Pantalla de formulario de creación de un punto de interés

En este caso, se ha ido con el mapa hasta San Fernando (Cádiz) y nos hemos posicionado sobre el *Castillo de San Romualdo*. La latitud y la longitud se calculan automáticamente con la posición del punto situado en la pantalla.

Ilustración 15: Pantalla resumen del punto que se va a crear

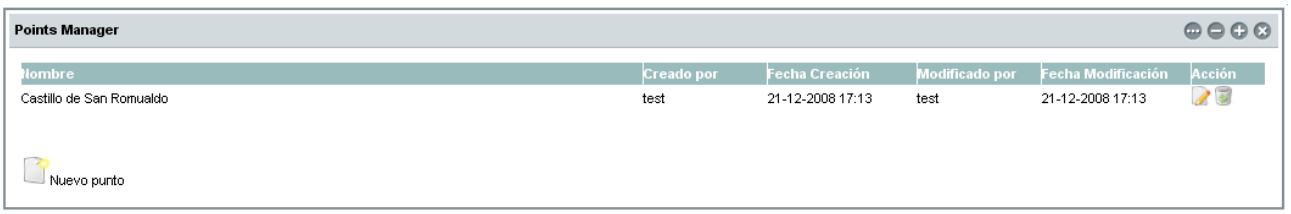

Ilustración 16: Pantalla de listado de puntos de interés

Una vez el punto ha sido creado, se creará un aviso de nuevo punto para que sea moderado. Los puntos creados, o modificados, por el usuario aparecerán en el listado y se podrán editar, o eliminar, a través de las acciones definidas a la derecha de la tabla.

Para terminar de completar el punto, accedemos a la acción de edición y le asignamos la categoría, el resumen y los detalles del punto de interés.

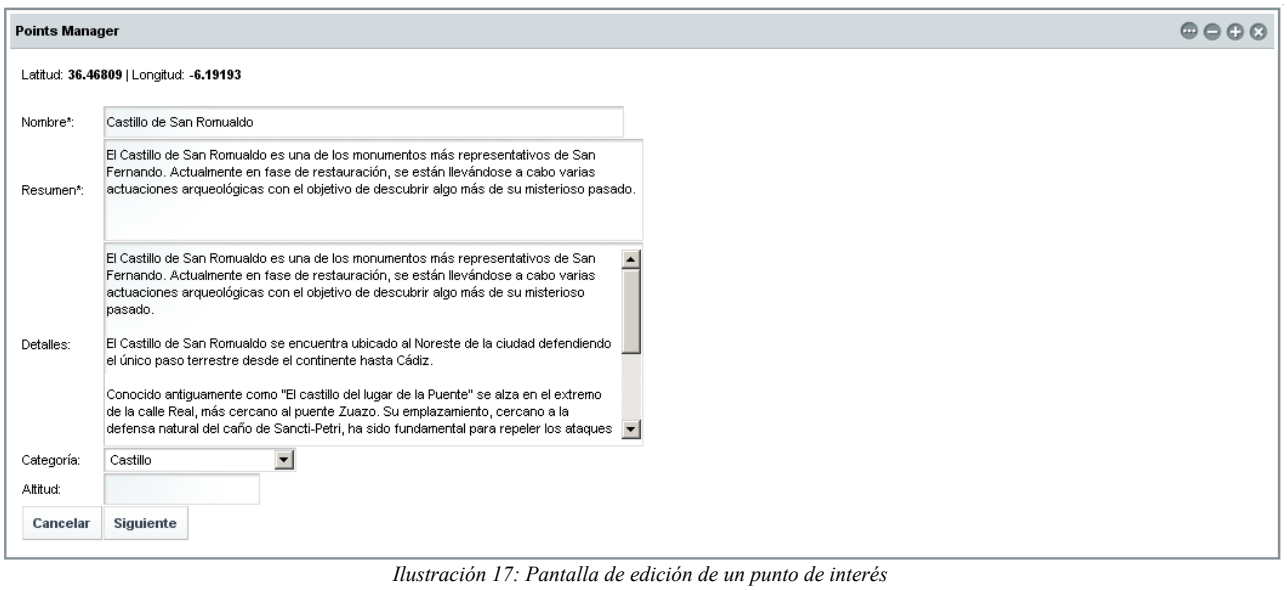

Ilustración 17: Pantalla de edición de un punto de interés

5.6.3 Points Moderator

Cuando se ha creado un punto de interés, se enviará creará una notificación de nuevo punto creado.

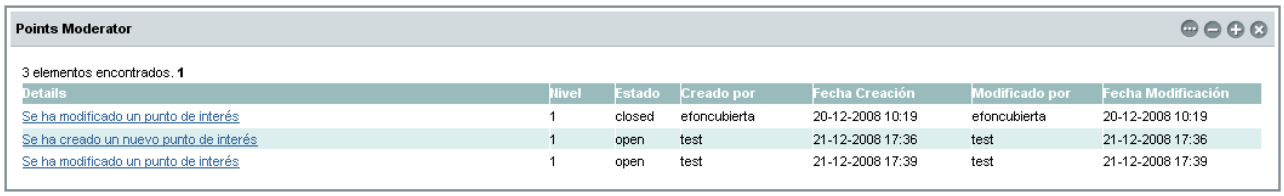

Ilustración 18: Pantalla de listado de avisos

En nuestro caso, y si estamos siguiendo la guía, tenemos que tener dos avisos. Un aviso por haber creado el punto y otro por haberlo modificado. Para habilitar el punto, accedemos a través del enlace del propio aviso.

Ilustración 19: Pantalla de visualización de aviso

6. CONCLUSIONES

El desarrollo de este proyecto ha servido para llevar a la práctica los conocimientos adquiridos durante el estudio de la carrera universitaria, especialmente las materias de Ingeniería del Software y de Metodología de la Programación.

Se han desarrollado diagramas UML mucho más complejos y completos que los que se diseñaron en la asignatura Ingeniería del Software, aunque se han utilizado menos tipos de diagramas al no creerse necesarios para elabora este proyecto.

Por otro lado, y hablando personalmente, ya tenía suficientes conocimientos en las tecnologías utilizadas excepto Spring, herramienta que había utilizado casualmente y que tenía en mente investigarla en profundidad a corto plazo. Con el desarrollo de este proyecto he conocido con mayor detalle el patrón *Dependency Inyection* y el contenedor *Inversion of Control*, y he quedado sorprendido de la potencia que ofrece la herramienta. Tanto es así que he centralizado todos los componentes del proyecto en Spring, con un resultado y una satisfacción excelentes, al ver delegadas y centralizadas las dependencias de los componentes en unos pocos ficheros XMLs.

También he aprendido cosas nuevas sobre Hibernate. Por un lado, integrar la herramienta en Spring ha sido muy sencillo gracias a la implementación de Hibernate en Spring en su módulo *Object-Relational Mapping (ORM)*. Solo ha sido necesario definir un DataSource, un Servicio de Hibernate e inyectarle a cada DAO dicho servicio. El resto ya se encarga de hacerlo Spring. Por otro lado, me encontré con una configuración de Hibernate un poco extraña al tener una clase *Node* y varias clases que heredan de él. El problema lo resolví utilizando *joinned subclasses*, que es un mecanismo que ofrece Hibernate para clases heredadas que se definen en diferentes tablas.

Struts lo he utilizado por conocer a fondo el framework, tanto en su versión 1 como en su nueva versión 2, que además puede ser usada con Spring, delegando así la definición de los Actions en Spring, creando un sistema mucho más homogeneo.

Finalmente, queda por decir que me hubiese gustado tener mucho más tiempo para desarrollar el proyecto, aunque por motivos laborales y el corto periodo de tiempo para el desarrollo del proyecto, he tenido que dejar muchas cosas en el tintero, como es desarrollar un motor de búsqueda. Desarrollar un buen motor sería tan complejo como un nuevo proyecto, ya que habría que avanzar en el uso de la API de Google Maps, además de diseñar algoritmos de acercamiento geográficos sobre las rutas calculadas. De todas formas, el proyecto deja abierta la posibilidad de inyectarle un nuevo motor de calculo a través de Spring. Tampoco se han desarrollado algunas pantallas para la gestión de la información, como puede ser la pantalla para crear comentarios, aunque toda la lógica de la capa de servicios de acceso a datos si está definida, además de las acciones de Struts que cargan esas pantallas. En general, al proyecto solo le faltan los siguientes desarrollos:

- ➔ Motor de cálculo de rutas: No se ha desarrollado por su complejidad, que bien podría ser tema de un único proyecto.

- ➔ Algunas pantallas: La lógica y las acciones si están realizadas, pero la pantalla JSP no.

Aun así, se ha cubierto todas las funcionalidades requeridas para el proyecto.

7. Bibliografía

- JSR-168 Portlet Specification [online]. The Java Community Process Program [27 Oct, 2003]. Disponible en Internet http://jcp.org/aboutJava/communityprocess/final/jsr168/index.html
- Spring Reference Manual v2.5.6 [online]. The Spring Framework [2008]. Disponible en Internet http://static.springframework.org/spring/docs/2.5.x/spring-reference.pdf
- Starting with Struts 2 [online]. Ian Roughley, InfoQ: C4Media [2006]. Disponible en Internet http://www.infoq.com/minibooks/starting-struts2. ISBN 978-1-4303-2033-3
- Struts 2 Documentation [online]. Apache Foundation [2006-2008]. Disponible en Internet http://struts.apache.org/2.x/docs/home.html
- Hibernate 3 Reference Manual [online]. Hibernate [2008]. Disponible en Internet http://www.hibernate.org/hib_docs/v3/reference/en-US/pdf/hibernate_reference.pdf
- Dependency Injection Pattern [online]. Unknow, Wikipedia [24 Jul, 2008]. Disponible en Internet http://en.wikipedia.org/wiki/Dependency_injection
- Inversion of Control [online]. Unknow, Wikipedia [10 Jun, 2008]. Disponible en Internet http://en.wikipedia.org/wiki/Inversion_of_control
- Data Access Object Pattern [online]. Sun Microsystems [2002]. Disponible en Internet http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html
- Model-View-Controller Pattern [online]. Unknow, Wikipedia [3 Dec, 2008]. Disponible en Internet http://en.wikipedia.org/wiki/Model-view-controller
- Liferay Administration Guide [online]. Richard L. Sevoz, Liferay Inc. [2008]. Disponible en Internet http://docs.liferay.com/portal/5.1/official/liferay-administration-guide-5.1.pdf. ISBN 978-0-615-24733-5
- Liferay Developer Guide [online]. Samuel Kong, Liferay Inc. [2008]. Disponible en Internet http://docs.liferay.com/portal/5.0/official/liferay-development-documentation-5.0.pdf