Como Trabajo de Fin de Grado (TFG) se ha realizado el análisis, diseño e implementación de una aplicación utilizando tecnología Java y una arquitectura Java EE.
Para llevar a cabo el desarrollo se han utilizado los conocimientos adquiridos a lo largo del Grado en Informática, en programación, análisis orientado a objetos y diseño de bases de datos. A su vez, ha sido necesario el estudio de la tecnología Java EE y de los diversos frameworks y componentes disponibles en el mercado.

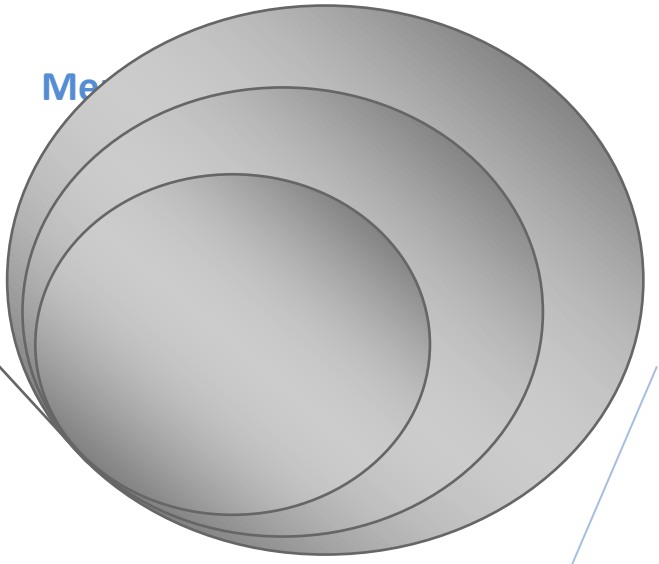
Área del TFG: Java EE

Palabras clave: Java, Java EE, JavaServer Faces, Ibatis, Patrones de diseño.

Licencia: © (Amaya García López)

Reservados todos los derechos. Está prohibida la reproducción total o parcial de esta obra por cualquier medio o procedimiento, incluidos la impresión, la reprografía, el microfilm, el tratamiento informático o cualquier otro sistema, así como la distribución de ejemplares mediante alquiler o préstamo, sin la autorización escrita del autor o de los límites que autorice la Ley de Propiedad Intelectual.

30 de diciembre de 2013

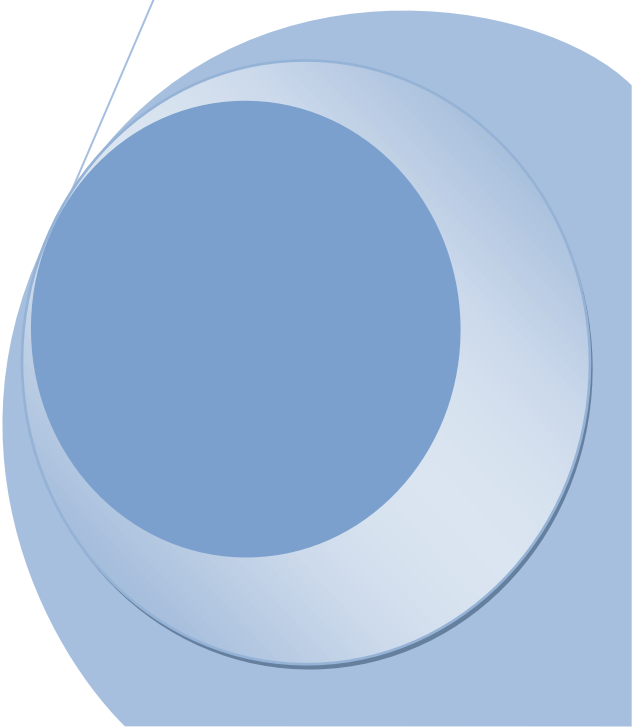


TFG JAVA EE: Sistema que gestiona las operaciones de un Agente de Seguros

Memoria

AMAYA GARCÍA LÓPEZ

30/12/2013



SÍNTESIS

Como Trabajo de Fin de Grado (TFG) se va a realizar el análisis, diseño e implementación de una aplicación utilizando tecnología Java y una arquitectura Java EE. El presente documento contiene la presentación y plan de trabajo a realizar durante el desarrollo de una aplicación que va a gestionar las tareas de un colaborador de seguros cuya dedicación es la intermediación entre el cliente y la compañía aseguradora. El proyecto de Seguros consiste en una aplicación accesible a través de Internet, en la que los colaboradores figura que será detallada posteriormente pueden gestionar un alta de póliza de seguros, alta de suplementos, consulta de póliza ,estado de la póliza, realizar gestiones cómo ver pólizas de sus clientes , etc.
Para llevar a cabo el desarrollo se han utilizado los conocimientos adquiridos a lo largo del Grado de Informática ,en programación, análisis orientado a objetos y diseño de bases de datos. A su vez, ha sido necesario el estudio de la tecnología Java EE y de los diversos frameworks y componentes disponibles en el mercado.

Área del TFG: Java EE

Palabras clave que se van a utilizar: Java, Java EE, JavaServer Faces, Ibatis, Patrones de diseño.

DESCRIPCIÓN DEL PROYECTO Y NECESIDADES A CUBRIR.

En los últimos años hemos vivido numerosos cambios en el sector asegurador, que van desde una gran diversificación de la oferta de seguros hasta la aparición de nuevos canales de distribución, pasando por un abaratamiento de precios y seguido de una reducción en la capacidad de suscribir riesgos industriales.

Tradicionalmente existían dos canales de distribución, que siguen manteniendo su importancia en la actualidad, a través de los cuales las entidades aseguradoras se dirigían al mercado. Estos dos canales de distribución son la venta directa, a través de la propia entidad o a través de agencias (agentes de seguros), y la distribución de seguros a través de mediadores independientes (corredores de seguros).

La transformación continua que se produce en el sector asegurador, fruto de la adaptación a un entorno cambiante y en constante evolución, repercute directamente en el consumidor de seguros, al que cada vez resulta más complejo conocer cuál es la mejor opción de seguro para satisfacer sus necesidades concretas.

En medio de todos estos cambios, la figura del mediador de seguros sigue manteniendo su importancia tradicional como distribuidor de seguros, si bien ha debido hacer un esfuerzo de adaptación a los cambios del mercado, cada vez más amplio y competitivo, lo que ha supuesto un aumento en la calidad de sus servicios, y es que el mediador de

seguros, como profesional independiente, imparcial y objetivo, sigue siendo el mejor asesor para el cliente o consumidor de seguros.

Cabe hacer una reflexión sobre el término "mediador de seguros", ya que se usa indistintamente para referirse a agentes y corredores de seguros, aunque en realidad se trata de dos figuras substancialmente distintas.

Agentes de Seguros

Los agentes de seguros (agentes o sociedades de agencia) son mediadores afectos a la entidad aseguradora, a los que une un vínculo contractual y una relación de dependencia, donde la entidad responde por las acciones del mediador y éste actúa en nombre de la entidad.

Se comprometen a realizar por cuenta de ésta labores de mediación, promoción, asesoramiento preparatorio y asistencia posterior con tomadores, asegurados o beneficiarios de los seguros.

Su vinculación debe ser exclusiva, es decir, no podrán estar vinculados simultáneamente por contrato de agencia con más de una entidad aseguradora, a menos que ésta le autorice expresamente para operar con otra entidad en determinados ramos, modalidades o contratos de seguros que no practique ella misma.

Las entidades aseguradoras llevan un registro de sus agentes, asignándoles un número de registro, dato que cada agente debe incluir en su publicidad y documentación propia.

Una vez descrita la figura del agente de seguros afecto a una compañía aseguradora y teniendo en cuenta la variabilidad organizativa existente en el mundo de las Compañías Aseguradoras , esta aplicación hace posible que un colaborador desde su casa con un teléfono-fax , una conexión a internet y una aplicación web , pueda gestionar su cartera de seguros .

Las necesidades que debe cubrir esta aplicación coinciden con las tareas propias de un Agente de Seguros afecto a una Compañía que serían:

GESTIÓN DE PÓLIZAS

- **Alta de un Presupuesto/Póliza** [1]
- **Alta de Suplementos**
- **Consulta de Presupuesto/Póliza**

GESTIÓN DE CLIENTES

- **Consulta de las pólizas de un cliente**
- **Informe del Cliente**

[1] Debemos de tener en cuenta que el sistema deberá validar si ese presupuesto es admitido (es decir cumple con unos criterios) y de este modo este presupuesto sería admitido. Y es entonces cuando se generará una póliza.
Esta tarea de generación de pólizas no es propia de un colaborador y no contemplada en este producto.

INFORMACIÓN

- **Centros médicos concertados (Asegurados de pólizas de accidente)**
- **Talleres concertados por ciudades**

Objetivos técnicos (uso de J2EE, diseño siguiendo patrones)

Requerimientos de software

La aplicación necesita que las siguientes aplicaciones estén instaladas previamente en el servidor:

Motor de base de datos MySql Community Server (http://www.mysql.com/)**:** La versión a utilizar la 5.5.9. Debe realizarse la instalación estándar y configurar la contraseña del usuario root. Luego es necesario ejecutar el script de creación de la base de datos .
Para administrar la base de datos voy a utilizar mysql-workbench-gpl-5.2.32-win32.

Java Enterprise Edition(http://java.sun.com/javaee/)**:**
Java Runtime Environment version "1.7.0_21-b11"
Java(TM) SE Runtime Environment (build 1.6.0_23-b05)
Java HotSpot(TM) Client VM (build 23.21-b01, mixed mode, sharing).
El proceso de instalación es el estándar.

Servidor web Apache Tomcat(http://tomcat.apache.org/)

JSP Standard Tag Library Apache Standard Taglib
(http://tomcat.apache.org/taglibs/)

Adicionalmente, el sistema necesita de un conjunto de librerías que no es necesario descargar ya que las mismas se proporcionan junto al proyecto:

Ibatis(http://www.mybatis.org/): Se utiliza para facilitar el mapeo objeto-relacional entre la base de datos y la aplicación Java.

Struts(http://struts.apache.org/):Es una herramienta de soporte para el desarrollo de aplicaciones Web bajo el patrón MVC bajo la plataforma Java EE (Java Enterprise Edition).

MySql Connector Java 5.1.15 (http://www.mysql.com/): Lo utiliza Ibatis para conectarse a la base de datos.

Apache Log4J 1.2.15 (http://logging.apache.org/log4j/1.2/): Es el componente elegido para dejar trazas de la ejecución del sistema.

Además de las herramientas propias del desarrollo J2EE se utilizarán también las siguientes:

Eclipse : Entorno de desarrollo integrado de código abierto.

Magic Draw UML : Es una herramienta CASE. La herramienta es compatible con el estándar UML 2.3, desarrollo de código para diversos lenguajes de programación Java.

OpenProj (http://sourceforge.net/projects/openproj/files/OpenProj%20Binaries/1.4/): Crea una completa planificación de tu proyecto.

Gimp , PhotoScape : Programas de edición de imágenes digitales en forma de mapa de bits, tanto dibujos como fotografías

Arquitectura de la aplicación

Teniendo en cuenta los requisitos iniciales, la arquitectura Java EE es la que mejor se adapta para el desarrollo del este proyecto, brindando una arquitectura multicapa y distribuida, accesible a través de Internet, al sistema.

Las capas lógicas de las que estará compuesta nuestra aplicación son las siguientes:

- Cliente: se conecta a Internet y accede a la aplicación desde un navegador.
- Vista: representa la interfaz visual con la que interactuará el usuario.
- Modelo: contiene las reglas de negocio del sistema.
- Persistencia: almacena y recupera la información de un motor de base de datos.

Entre las principales ventajas de utilizar esta arquitectura multicapa encontramos:

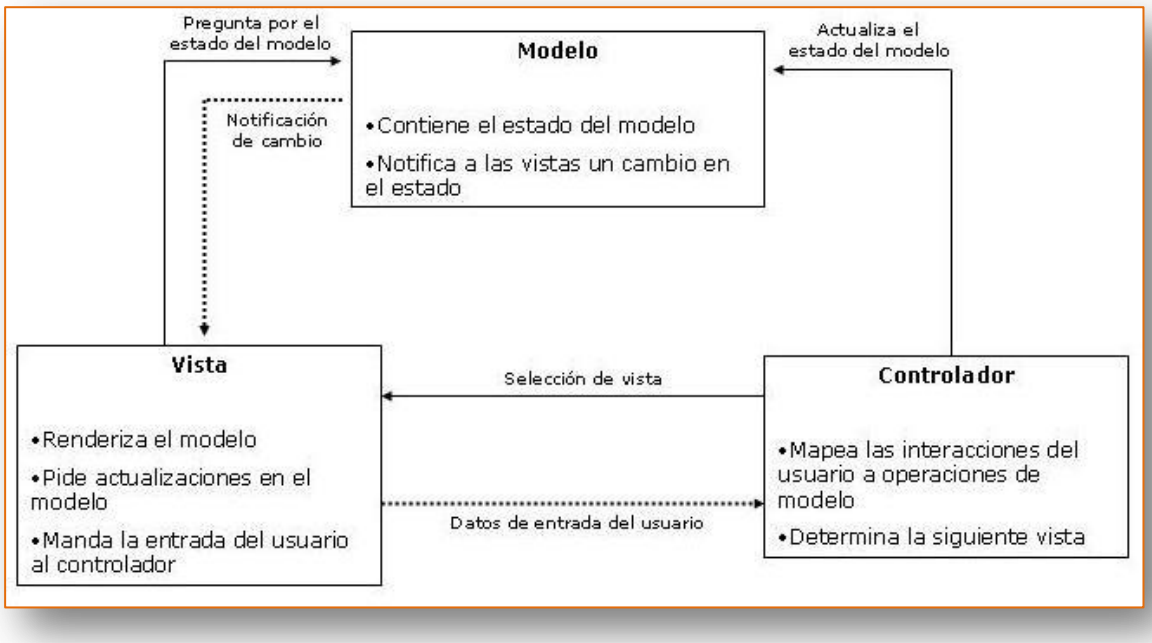
- Mantenimiento: una modificación en una capa no implica la modificación del resto de capas de la aplicación.

- Reusabilidad: Los datos y el modelo sólo se definen una vez, permitiendo a otras aplicaciones utilizarlos sin violar las reglas del sistema.

- Escalabilidad: el sistema es flexible para dividirse físicamente cuando los requerimientos sobre el desempeño de la aplicación cambian.

- Cliente ligero: Los usuarios pueden conectarse a la aplicación sin importar el sistema operativo que utilizan y beneficiarse de mejoras en el sistema sin necesidad de descargar ningún software adicional.

- Hay que tener en cuenta que para facilitar el desacoplamiento entre la Vista y el Modelo se utiliza el patrón de diseño **Modelo-Vista-Controlador (MVC).** Al utilizar MVC, los datos de negocio están separados de la lógica de presentación, y



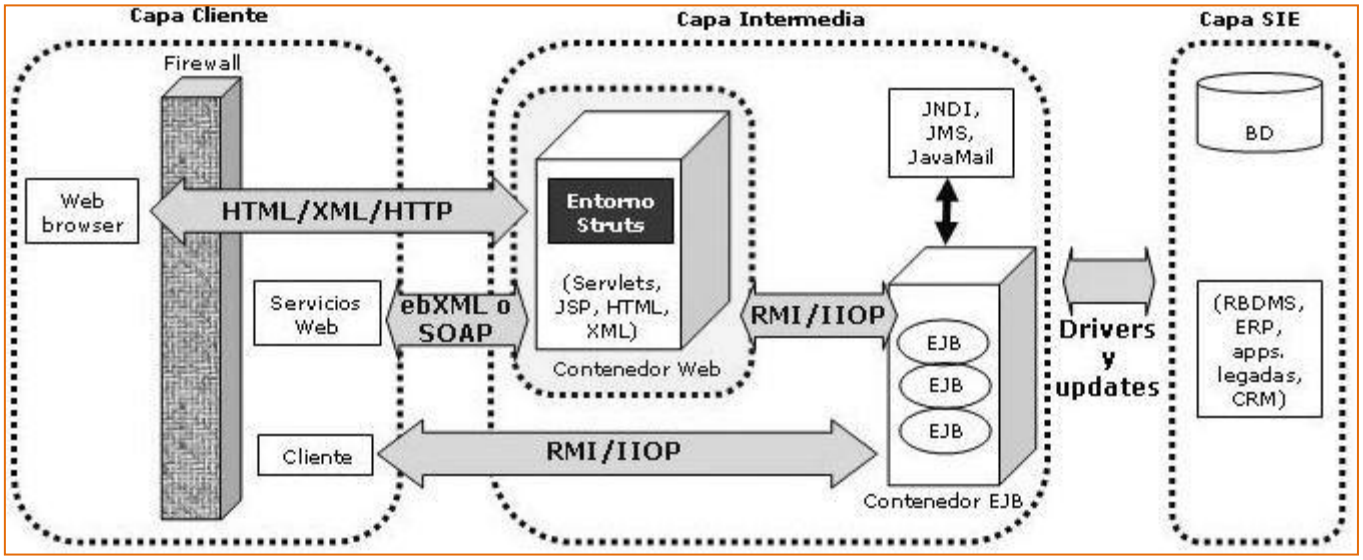
los componentes que procesan los datos no controlan la visualización de los mismos y viceversa.

Para la implementación física de esta arquitectura se han tomado las siguientes decisiones que afectan a la fase de desarrollo:

- Java: Java Runtime Environment version "1.7.0_21".

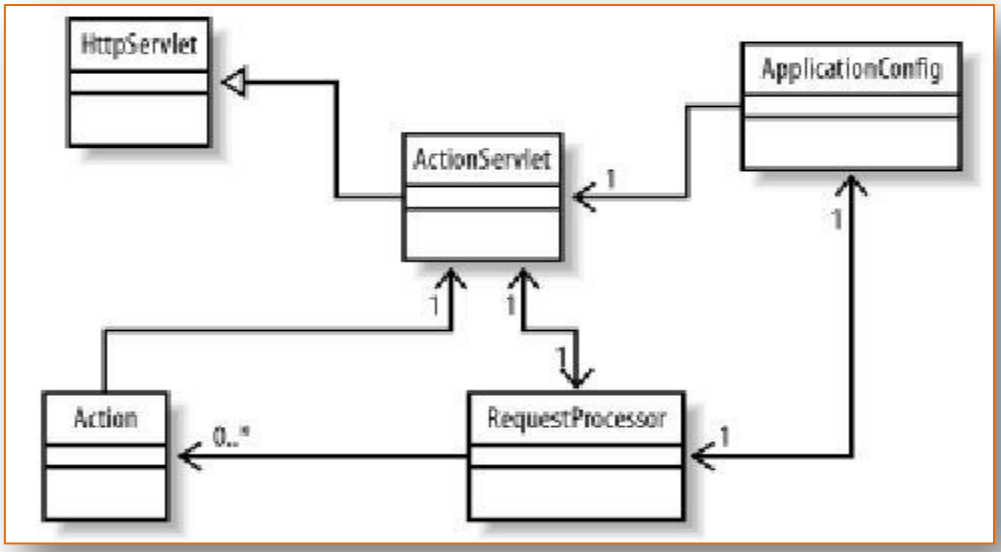
- Presentación: **JavaServer Pages** (**JSP**) para generar las páginas web de la aplicación e implementar el controlador de la arquitectura MVC.

- **Struts** es una herramienta de soporte para el desarrollo de aplicaciones Web bajo el patrón MVC bajo la plataforma Java EE (Java Enterprise Edition).



Las aplicaciones Struts residen en un contenedor Web (dentro de un servidor de aplicaciones) y pueden hacer uso de los servicios proporcionados por el contenedor, como el manejo de peticiones HTTP o HTTPS. Esto permite al desarrollador olvidarse de la lógica de negocio. Así, Struts hace uso de varias áreas de recursos compartidos para almacenar objetos: petición (*javax.servlet.http.HttpServletRequest*), sesión (*javax.servlet.http.HttpSession*), aplicación (*javax.servlet.ServletContext*) y página (*javax.servlet.jsp.PageContext*).

Desde el punto de vista de la arquitectura MVC, las clases que proporciona Struts respecto de la capa C son:



- **ActionServlet**, clase que extiende javax.servlet.http.HttpServlet responsable del empaquetado y enrutado del tráfico HTTP hacia el manejador apropiado dentro del entorno (framework). No es abstracta, y por tanto se puede utilizar directamente en cualquier aplicación.

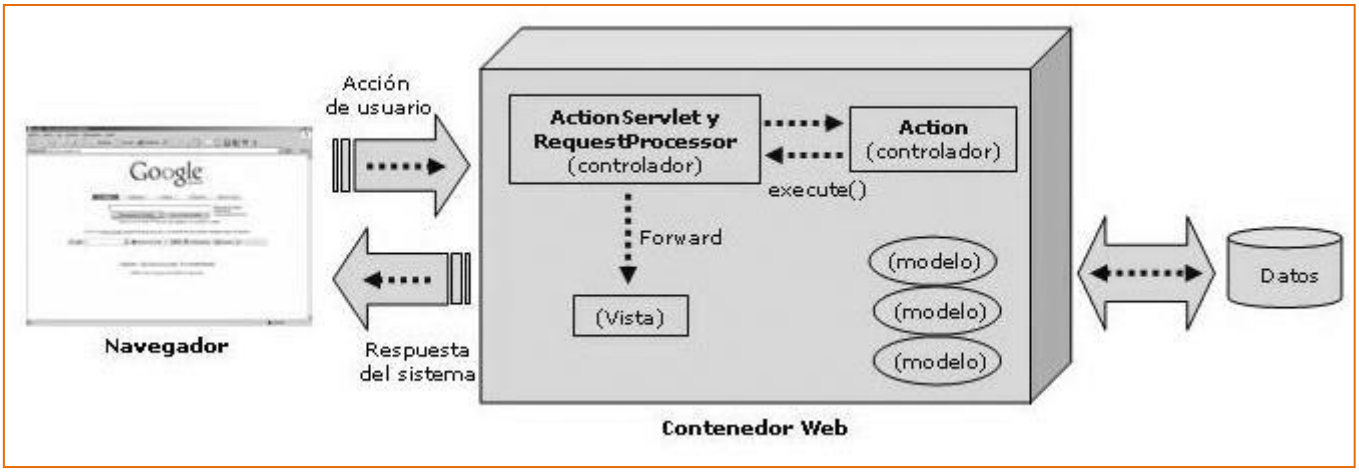
- **RequestProcessor**, clase que permite desacoplar el proceso de petición (request process) del ActionServlet y así poder modificar cómo se procesa la petición (haciendo un 'subclass' de RequestProcessor).

- **Action**, clase que independiza la petición del cliente del modelo de negocio. Es una extensión del componente de control (capa C) y permite llevar a cabo funciones como autorización, logging, o validación de sesión, antes de invocar la lógica de negocio. Su método más importante es:

public ActionForward execute(ActionMapping mapping, HttpServletRequest request, HttpServletResponse response) throws Exception;

- **ActionMapping**, clase que representa una acción de mapeado que se define en el fichero de configuración de Struts. Indica al controlador que instancia de Action se debe invocar en cada petición.

- **ActionForward**, clase que representa el destino al cual el controlador debe enviar el control una vez que una Action se ha completado (nos evitamos meter el 'forward' en la página JSP).

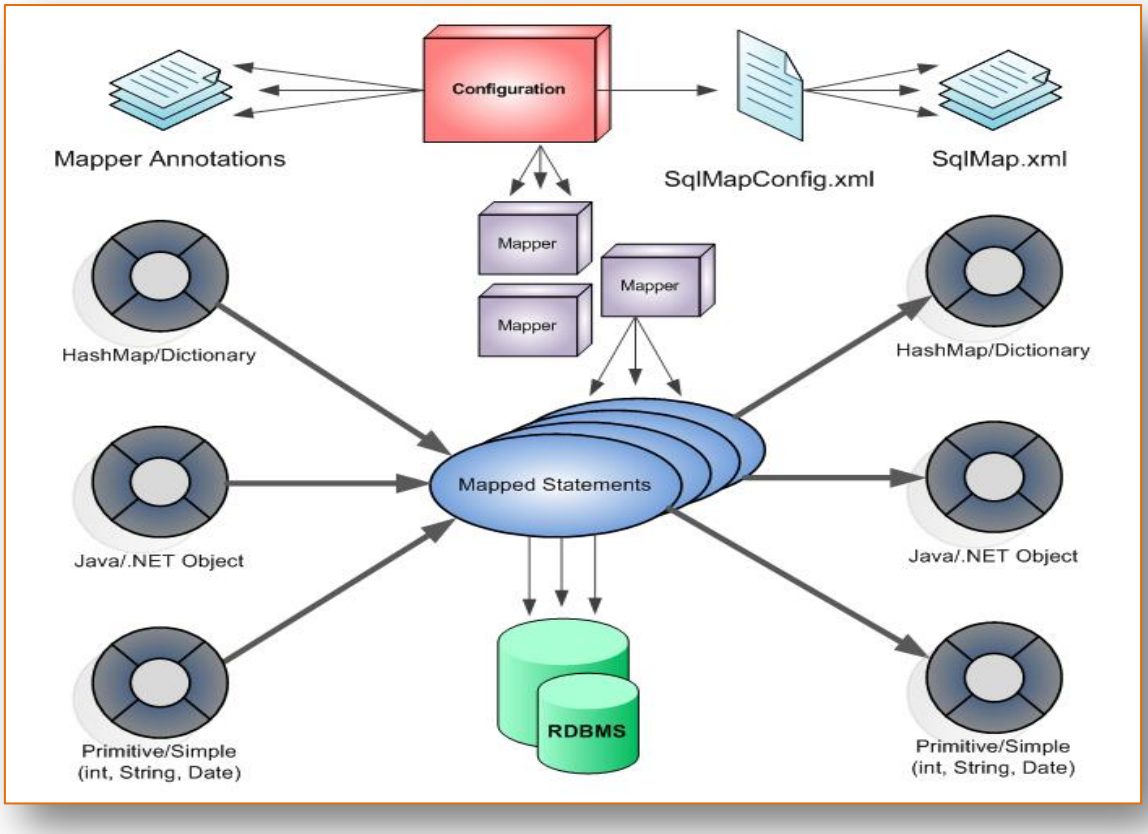


Con respecto de la capa M, Struts no fue diseñado para trabajar con esta capa. Sin embargo, es obvio que Struts recibirá información de esta capa (aunque no sepa cómo está implementada). Así, la mejor forma de solucionar cómo Struts se comunica con la capa de negocio de forma que sea completamente independiente de la tecnología de implementación de ésta, es utilizar un patrón de diseño, como por ejemplo DTOs (Data Transfer Objects), también llamados Value Objects (VO).
La idea es que Struts reciba la información en forma de vistas (VOs), pero no sepa cómo se han creado esas vistas. Para ello también necesitaremos implementar un nuevo patrón de diseño muy conocido, el Business Delegate. Con este patrón, crearemos un servicio perteneciente a la capa de negocio, que servirá de nexo de unión con la capa de control y será a través del cual Struts pedirá y recibirá los únicos objetos que entiende y con los que sabe trabajar: los VOs. Por el otro lado, la capa de negocio sólo sabrá que puede recibir peticiones del servicio de acceso a BD a través de JDBC y devolverá VOs, sin importarle ni quién los pide ni quién los usa .

Por último, con respecto a la capa V, los elementos de los que hace uso Struts son:

- HTML
- Value Objects (vistas)
- **ActionForms**, clase que permite pasar datos de entrada hacia delante y hacia atrás entre el usuario y la capa de negocio.
- JavaServer Pages
- Struts Tags (librería de etiquetas)
- Otros recursos Java

❖ Persistencia: Se utiliza **MySql** debido a que se trata de un motor de base de datos gratuito y extensamente utilizado e **Ibatis.** La ventaja de utilizar Ibatis frente a JDBC es que nos permite mapear las relaciones y atributos de nuestras tablas en objetos e independizar el código de nuestra aplicación de un cambio de la base de datos.



- Servidor web**: <u>Dado que no utilizaré EJB</u>**, se utiliza como servidor web **Apache Tomcat** ya que es gratuito y de los más usados. Este servidor implementa las especificaciones de servlets y JSP que necesitaremos para ofrecer el acceso a la aplicación a través de internet.

Planificación con los hitos principales (etapa de análisis, etapa de diseño, etapa de implementación)

Tareas a desarrollar . Una vez analizada la descripción del proyecto podemos asumir que los requisitos no cambiarán durante la realización del TFG . Usando un ciclo de vida en cascada clásico podemos prever las siguientes tareas a desarrollar.

Documento a entregar[2]	Fase	Tarea	Duración estimada
PEC 1	Plan de trabajo	Definición de los requerimientos	2 días
		Definición de los objetivos	3 día
		Planificación del proyecto	1 día
PEC 2	Análisis	Descripción de casos de uso y actores	4 días
		Prototipo	3 días
		Glosario	1 día
	Diseño	Revisión del análisis	3 días
		Diagrama de clases	5 días
		Diagramas de secuencia, estado y actividad	6 días
		Diseño Relacional de BD	4 días
		Diseño de la arquitectura	10 días
PEC 3	Implementación	Instalación del Entorno de trabajo	3 días
		Creación de la BD	3 días
		Programación	28 días
		Pruebas	3 días
		Manual de usuario	2 días
Entrega Final	Memoria Presentación Producto	Presentación	7 días
		Memoria	9 días
		Producto final (Repaso general de todo el proyecto)	12 días

[2] Planificación del proyecto construida con OpenProj.

	Nombre	Duración	Inicio	Terminado
1	**TFG-Java EE**	**111 days**	**20/09/13 8:00**	**8/01/14 17:00**
2	**PEC 1- PLAN DE TRABAJO**	**6 days**	**20/09/13 8:00**	**25/09/13 17:00**
3	**Plan de trabajo**	**6 days**	**20/09/13 8:00**	**25/09/13 17:00**
4	Requerimientos	2 days	20/09/13 8:00	21/09/13 17:00
5	Objetivos	3 days	22/09/13 8:00	24/09/13 17:00
6	Planificación	1 day	25/09/13 8:00	25/09/13 17:00
7	**PEC2-ANÁLISISYDISEÑO**	**36 days**	**26/09/13 8:00**	**31/10/13 17:00**
8	**ANÁLISIS**	**8 days**	**26/09/13 8:00**	**3/10/13 17:00**
9	Descripción de casos de uso y actores	4 days	26/09/13 8:00	29/09/13 17:00
10	Prototipo	3 days	30/09/13 8:00	2/10/13 17:00
11	Glosario	1 day	3/10/13 8:00	3/10/13 17:00
12	**DISEÑO**	**28 days**	**4/10/13 8:00**	**31/10/13 17:00**
13	Revisión del análisis	3 days	4/10/13 8:00	6/10/13 17:00
14	Diagrama de clases	5 days	7/10/13 8:00	11/10/13 17:00
15	Diagramas de secuencia, estado y actividad	6 days	12/10/13 8:00	17/10/13 17:00
16	Diseño Relacional de BD	4 days	18/10/13 8:00	21/10/13 17:00
17	Diseño de la arquitectura	10 days	22/10/13 8:00	31/10/13 17:00
18	**PEC3-IMPLEMENTACIÓN**	**39 days**	**1/11/13 8:00**	**9/12/13 17:00**
19	**IMPLEMENTACIÓN**	**39 days**	**1/11/13 8:00**	**9/12/13 17:00**
20	Instalación del Entorno de trabajo	3 days	1/11/13 8:00	3/11/13 17:00
21	Creación de la BD	3 days	4/11/13 8:00	6/11/13 17:00
22	Programación	28 days	7/11/13 8:00	4/12/13 17:00
23	Pruebas	3 days	5/12/13 8:00	7/12/13 17:00
24	Manual de usuario	2 days	8/12/13 8:00	9/12/13 17:00
25	**ENTREGA FINAL**	**30 days**	**10/12/13 8:00**	**8/01/14 17:00**
26	Presentación	7 days	10/12/13 8:00	16/12/13 17:00
27	Memoria	9 days	16/12/13 8:00	24/12/13 17:00
28	Producto Final	12 days	28/12/13 8:00	8/01/14 17:00

3

[3] **Planificación hecha con *OPENPROJ* para PEC 1**

1. Análisis

En este apartado avanzaremos en el análisis del problema propuesto, describiendo los actores y los casos de uso más importantes. También se presentará un prototipo del sitio web a desarrollar y un glosario.

1.1. Actores

Los actores que interactúan con la aplicación **Sistema que gestiona las operaciones de un Agente de Seguros** son:

Colaborador: Son prácticamente los usuarios de la aplicación y pueden realizar las siguientes acciones:

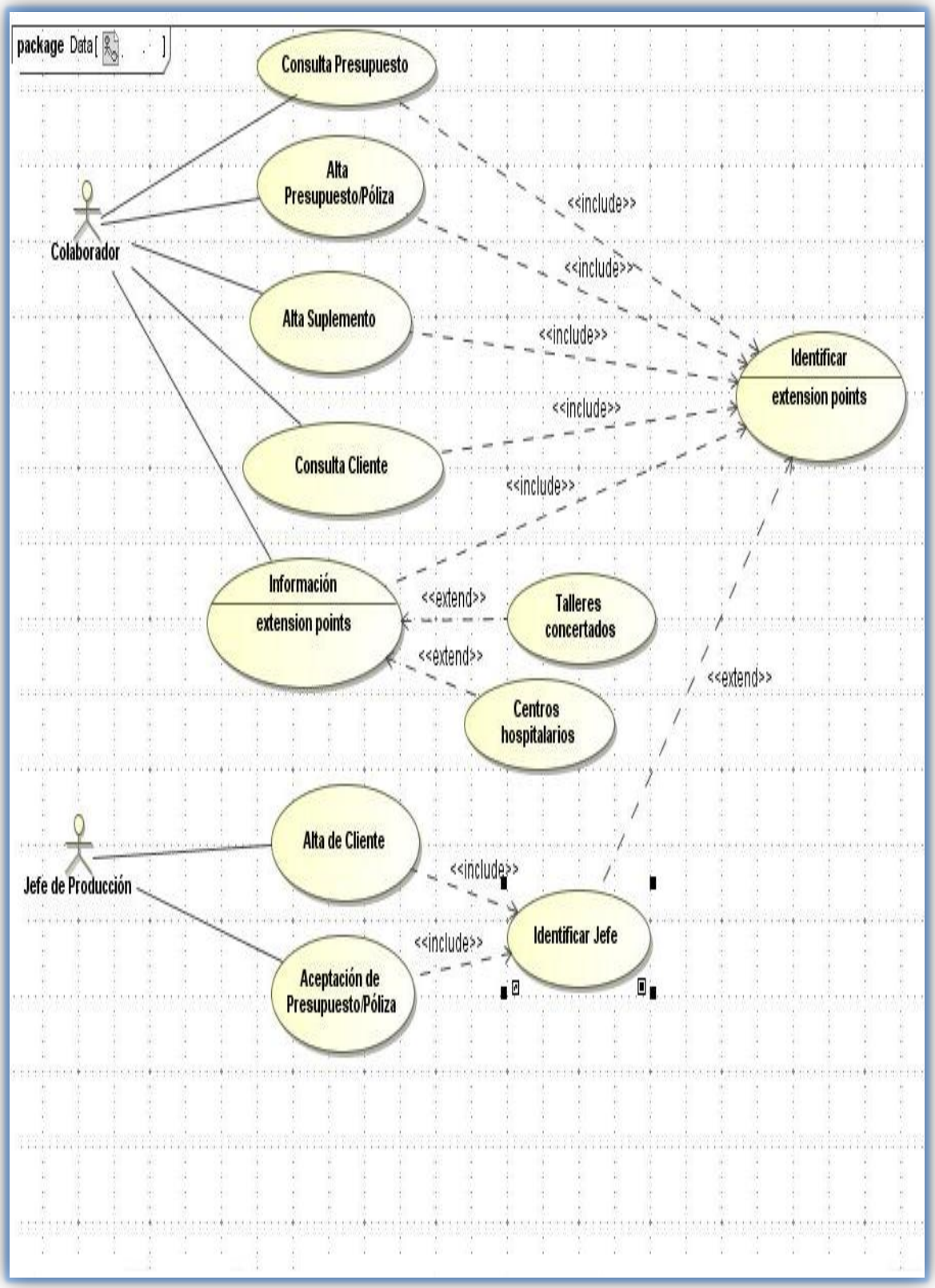
- Identificarse en la aplicación
- Alta de Presupuesto/Póliza
- Alta de suplemento
- Consulta de clientes
- Ver información sobre talleres concertados y centros hospitalarios
- Consulta de Presupuesto/Póliza

Jefe de producción: Son usuarios que deben ratificar los datos que el usuario colaborador ha introducido, además de las siguientes tareas:

- Identificarse en la aplicación
- Alta de cliente
- Aceptación Presupuesto/Póliza

1.2. Casos de uso

De la descripción del proyecto podemos identificar los siguientes casos de uso:


package Data [. .]
Consulta Presupuesto
Alta Presupuesto/Póliza
Colaborador
Alta Suplemento
Consulta Cliente
Información
extension points
Identificar
extension points
Talleres concertados
Centros hospitalarios
Jefe de Producción
Alta de Cliente
Identificar Jefe
Aceptación de Presupuesto/Póliza
<<include>>
<<include>>
<<include>>
<<include>>
<<include>>
<<extend>>
<<extend>>
<<extend>>
<<include>>
<<include>>

1.3. Descripción textual de los casos de uso

1. Caso de uso **Identificarse**
1.1. Actor principal: **Colaborador**
1.2. Precondición: Ninguna
1.3. Postcondición: El usuario está identificado
1.4. Casos de uso relacionados: **Consulta clientes, Alta Póliza/Presupuesto, Alta Suplemento, Consulta Presupuestos, Consultar información, Aceptación Póliza, Alta Cliente.**
2. Escenario principal:
2.1. El usuario indica su código de usuario y contraseña.
2.2. El sistema verifica que se trata de un usuario válido
3. Flujos alternativos:
3.1. Si el usuario no existe en el sistema o la contraseña es incorrecta muestra un error.

1. Caso de uso **Alta Póliza/Presupuesto**
1.1. Actor principal: **Colaborador**
1.2. Precondición: El usuario se ha identificado previamente
1.3. Postcondición: Ninguna
2. Escenario principal:
2.1. El usuario se conecta a la aplicación.
2.2. El sistema muestra un formulario que contiene todos los datos del Presupuesto.
2.3. El colaborador rellena todos los datos, teniendo en cuenta que son obligatorios para poder validar el alta**.**
2.4 Buscamos bonificación si existiera para este cliente, nos conectamos a la Base de datos para obtener este dato.
3. Flujos alternativos:
3.1. El sistema busca en BD **la bonificación** si existiese para el cliente que nos ocupa**.**

1. Caso de uso **Alta Suplemento**
1.1. Actor principal: **Colaborador**
1.2. Precondición: El usuario se ha identificado previamente, Debe existir la póliza a la que se le añade suplemento.
1.3. Postcondición: Ninguna
2. Escenario principal:
2.1. El usuario se conecta a la aplicación.
2.2. El sistema muestra un formulario que contiene todos los datos del Suplemento.
2.3. El colaborador rellena todos los datos, teniendo en cuenta que son obligatorios para poder validar el alta**.**
3. Flujos alternativos:
3.1. Si la póliza no existe en el sistema muestra un error y no permite seguir.

1. Caso de uso **Ver Presupuesto**
1.1. Actor principal: **Colaborador**
1.2. Precondición: El usuario se ha identificado previamente
1.3. Postcondición: Ninguna
2. Escenario principal:
2.1. El sistema muestra una lista de los presupuestos, mostrando el número de póliza, datos de tomador del seguro, la fecha de efecto, las garantías solicitadas, estado del presupuesto (pendiente o aceptado).
3. Flujos alternativos:
3.1. El usuario puede buscar otras pólizas de este cliente.

1. Caso de uso **Ver información**
1.1. Actor principal: **Colaborador**
1.2. Precondición: El usuario se ha identificado previamente
1.3. Postcondición: Ninguna
2. Escenario principal:
2.1. El usuario solicita ver información sobre talleres concertados (póliza de auto) y de centros hospitalarios (póliza de accidentes y hospitalización)
3. Flujos alternativos:
3.1. El usuario selecciona un taller u hospital si fuera el caso para ver su detalle.

1. Caso de uso **Consulta Cliente**
1.1. Actor principal: **Colaborador**
1.2. Precondición: El usuario se ha identificado previamente
1.3. Postcondición: Ninguna
1.4. Casos de uso relacionados: **Identificarse, Buscar Cliente**
2. Escenario principal:
2.1. El sistema muestra un informe del cliente solicitado en la búsqueda , mostrando datos personales , pólizas asociadas a éste.
3. Flujos alternativos:
3.1. El usuario antes de ver los datos del cliente debe filtrar de qué cliente quiere recibir información.

1. Caso de uso **Alta Cliente**
1.1. Actor principal: **Jefe de producción**
1.2. Precondición: El usuario se ha identificado previamente, Debe poseer alguna póliza
1.3. Postcondición: Ninguna
2. Escenario principal:
2.1. El usuario se conecta a la aplicación.
2.2. El sistema muestra un formulario que contiene todos los datos del Cliente.

2.3. El colaborador rellena todos los datos, teniendo en cuenta que son obligatorios para poder validar el alta**.**

3. Flujos alternativos:

3.1. El sistema busca en BD si existiesen pólizas para el cliente que nos ocupa**.**

1. Caso de uso **Aceptación de presupuesto**

1.1. Actor principal: **Jefe de producción**

1.2. Precondición: El usuario se ha identificado previamente , Debe existir un alta de presupuesto

1.3. Postcondición: Ninguna

2. Escenario principal:

2.1. El usuario se conecta a la aplicación.

2.2. El sistema muestra los datos que se han de valorar desde la lógica del negocio para poder aceptar la póliza.

2.4 Se estudia si tanto el cliente como las garantías exigidas por éste cumplen con lo establecido, si no fuera así ésta se rechaza.

3. Flujos alternativos:

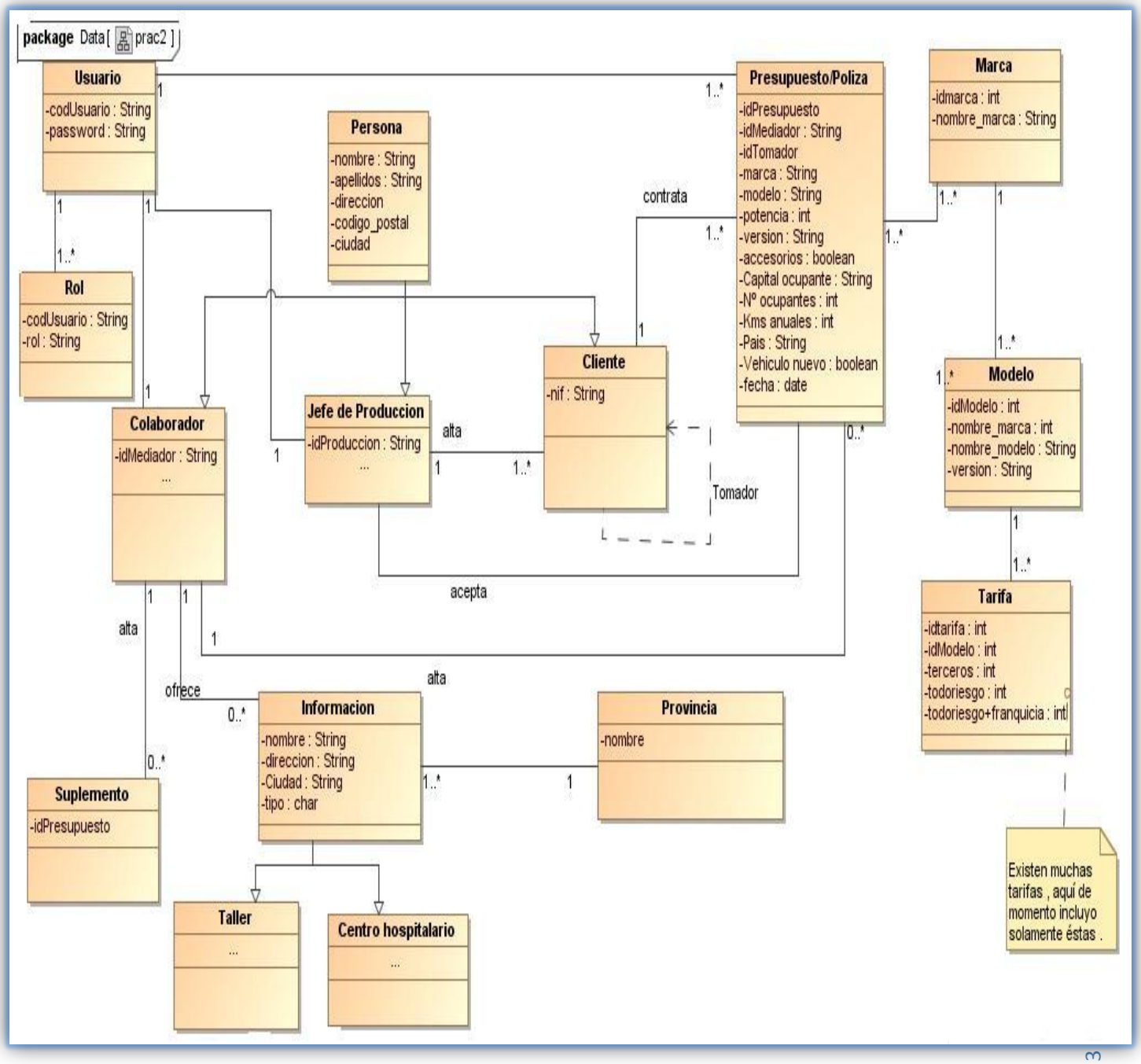
3.1. El sistema busca puede rechazar una póliza mostrando las razones por las que esa póliza no es aceptada**.**

2. Diseño

Una vez acotado el problema en la fase de análisis, continuamos con el diseño de la aplicación Gestión de Seguros presentando los diagramas de clases, estado, actividad y secuencia más significativos. También se debe realizar el diseño de la base de datos y definir la arquitectura a utilizar.

2.1. Diagrama de clases

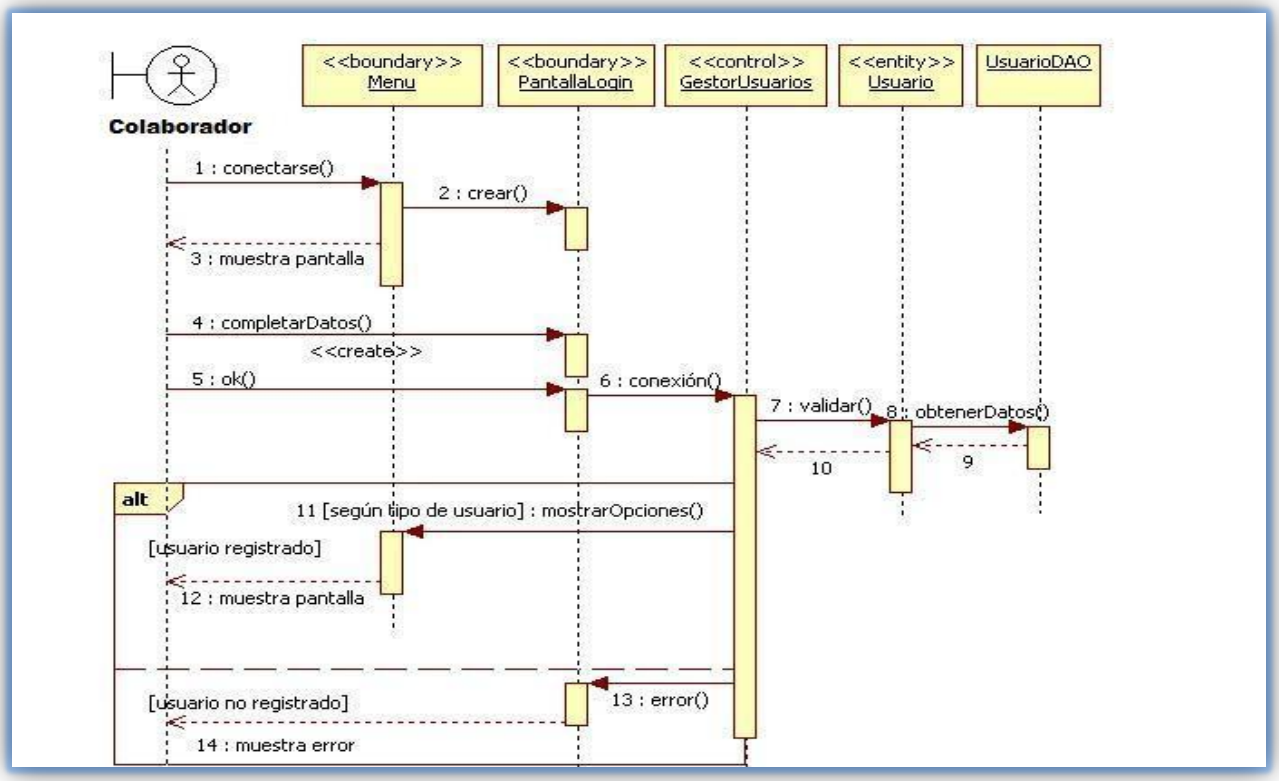
A continuación se presenta el diagrama de clases de entidad junto con sus atributos principales:


package Data [prac2]
Usuario
-codUsuario : String
-password : String
Persona
-nombre : String
-apellidos : String
-direccion
-codigo_postal
-ciudad
Presupuesto/Poliza
-idPresupuesto
-idMediador : String
-idTomador
-marca : String
-modelo : String
-potencia : int
-version : String
-accesorios : boolean
-Capital ocupante : String
-Nº ocupantes : int
-Kms anuales : int
-Pais : String
-Vehiculo nuevo : boolean
-fecha : date
Marca
-idmarca : int
-nombre_marca : String
contrata
Rol
-codUsuario : String
-rol : String
Cliente
-nif : String
Modelo
-idModelo : int
-nombre_marca : int
-nombre_modelo : String
-version : String
Colaborador
-idMediador : String
...
Jefe de Produccion
-idProduccion : String
...
alta
Tomador
acepta
Tarifa
-idtarifa : int
-idModelo : int
-terceros : int
-todoriesgo : int
-todoriesgo+franquicia : int
alta
ofrece
Informacion
-nombre : String
-direccion : String
-Ciudad : String
-tipo : char
Provincia
-nombre
Suplemento
-idPresupuesto
Taller
...
Centro hospitalario
...
Existen muchas tarifas , aquí de momento incluyo solamente éstas .

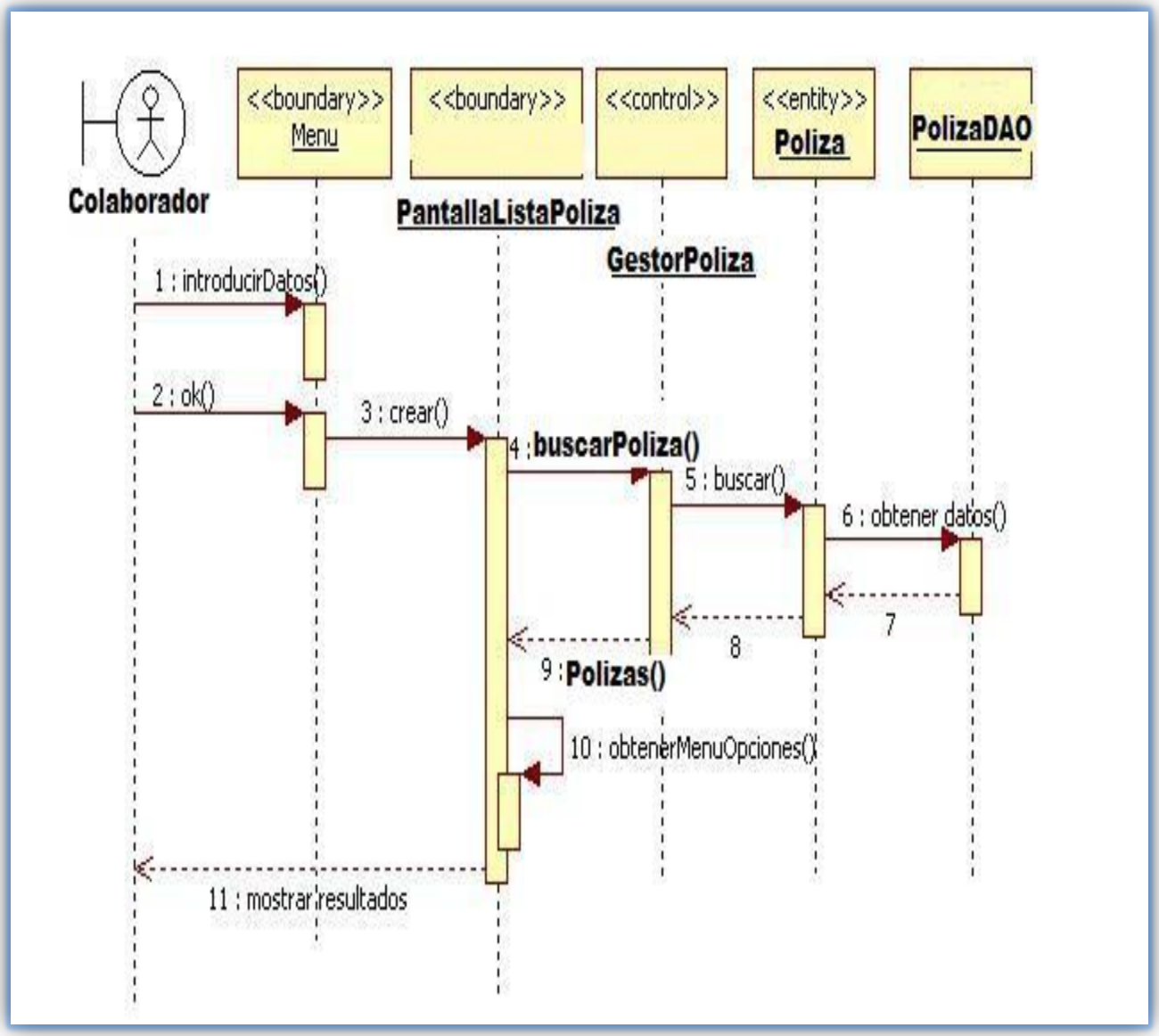
2.2. Diagramas de secuencia

Continuando con el modelado de los aspectos dinámicos del sistema, se presentan los diagramas de secuencia para algunos de los casos de uso más representativos de la aplicación.
Hay que tener en cuenta que es posible que en la etapa de implementación aparezcan otras colaboraciones de clase no representadas por estos diagramas. Sin embargo, estos siguen siendo válidos en tanto presentan una primera visión de las interacciones entre clases que sucederán dentro de la aplicación.

Identificación en el sistema:

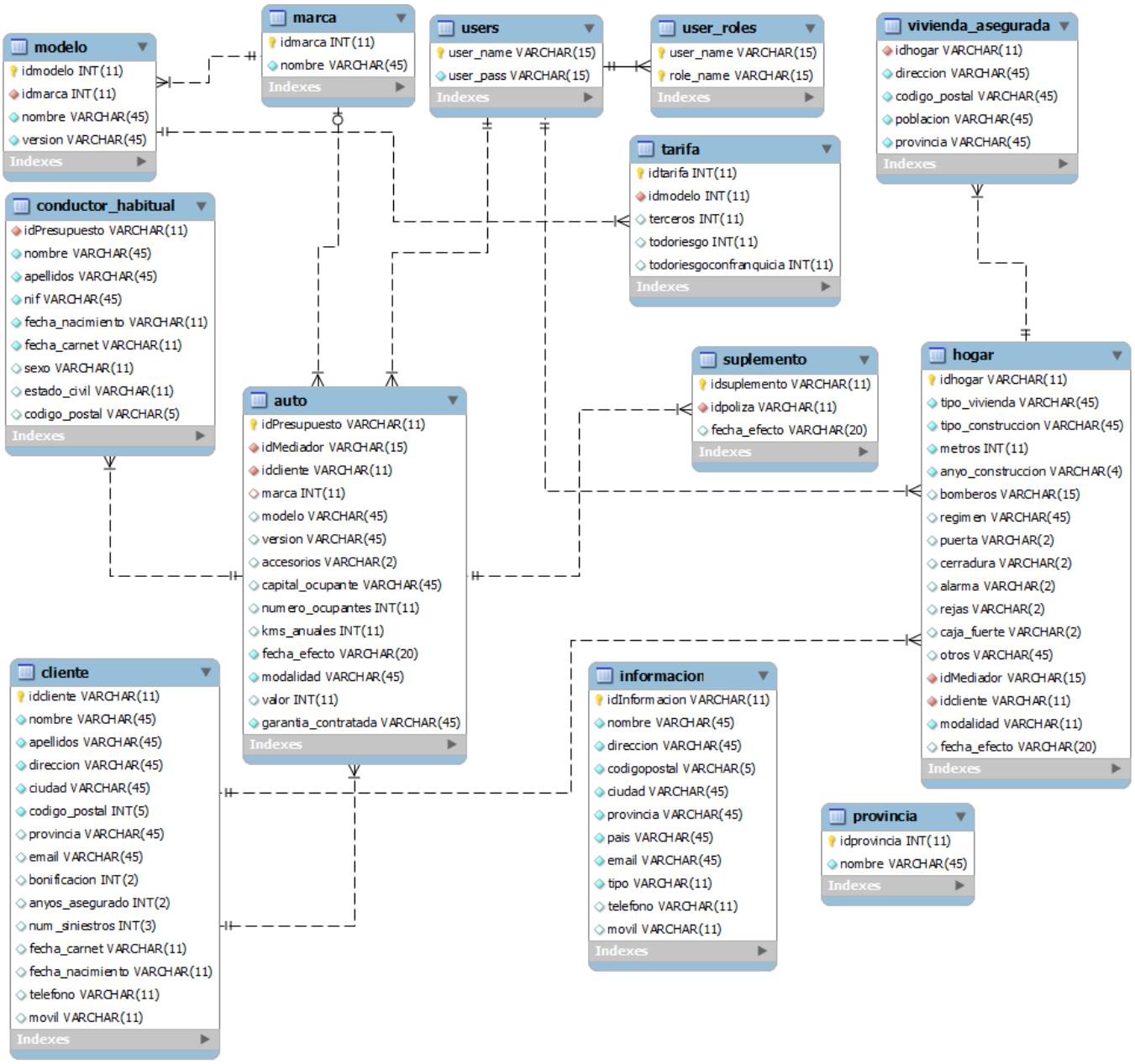


Consulta de Pólizas:



2.3. Diseño de la base de datos

Utilizando un modelo EER para el diseño de la base de datos, obtenemos el modelo relacional siguiente:



3. Implementación

Finalizada la fase de implementación, se explican las principales decisiones tomadas durante el desarrollo y los requerimientos de software del sistema. También se indican los pasos a realizar para una correcta configuración y ejecución de la aplicación.

Todos los casos de uso han sido implementados en esta etapa, de modo que la aplicación desarrollada es completamente funcional y cumple con la mayoría de los requisitos especificados al inicio del proyecto.

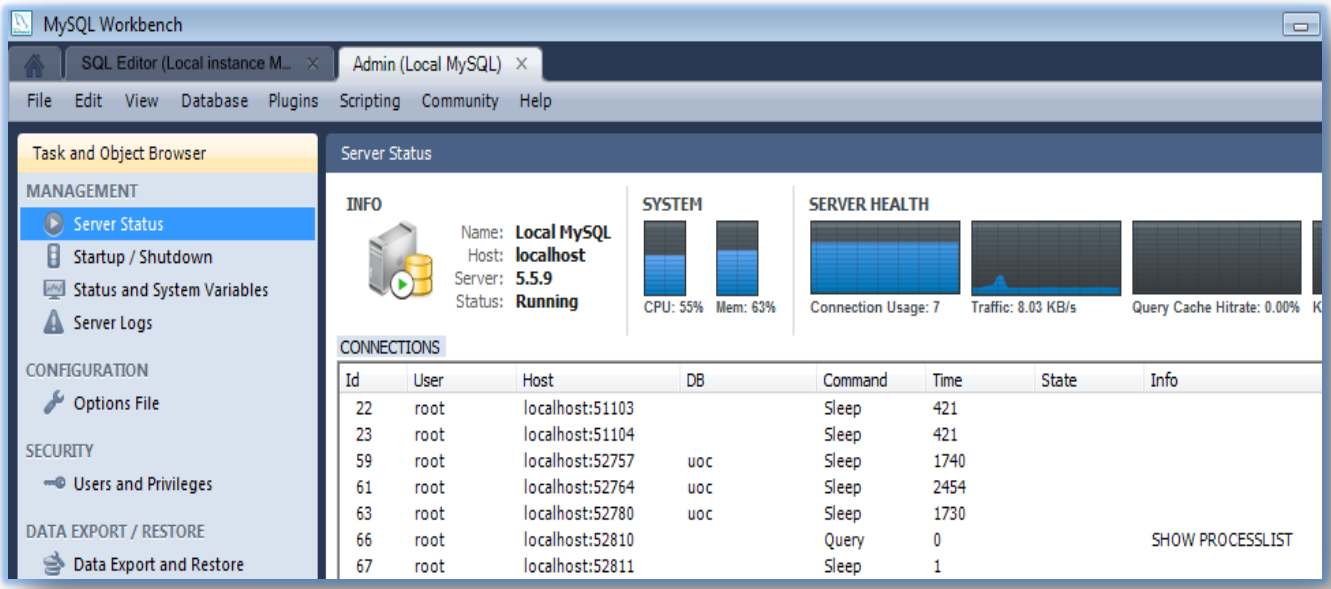
3.1. Requerimientos de software

La aplicación necesita que las siguientes aplicaciones estén instaladas previamente en el servidor:

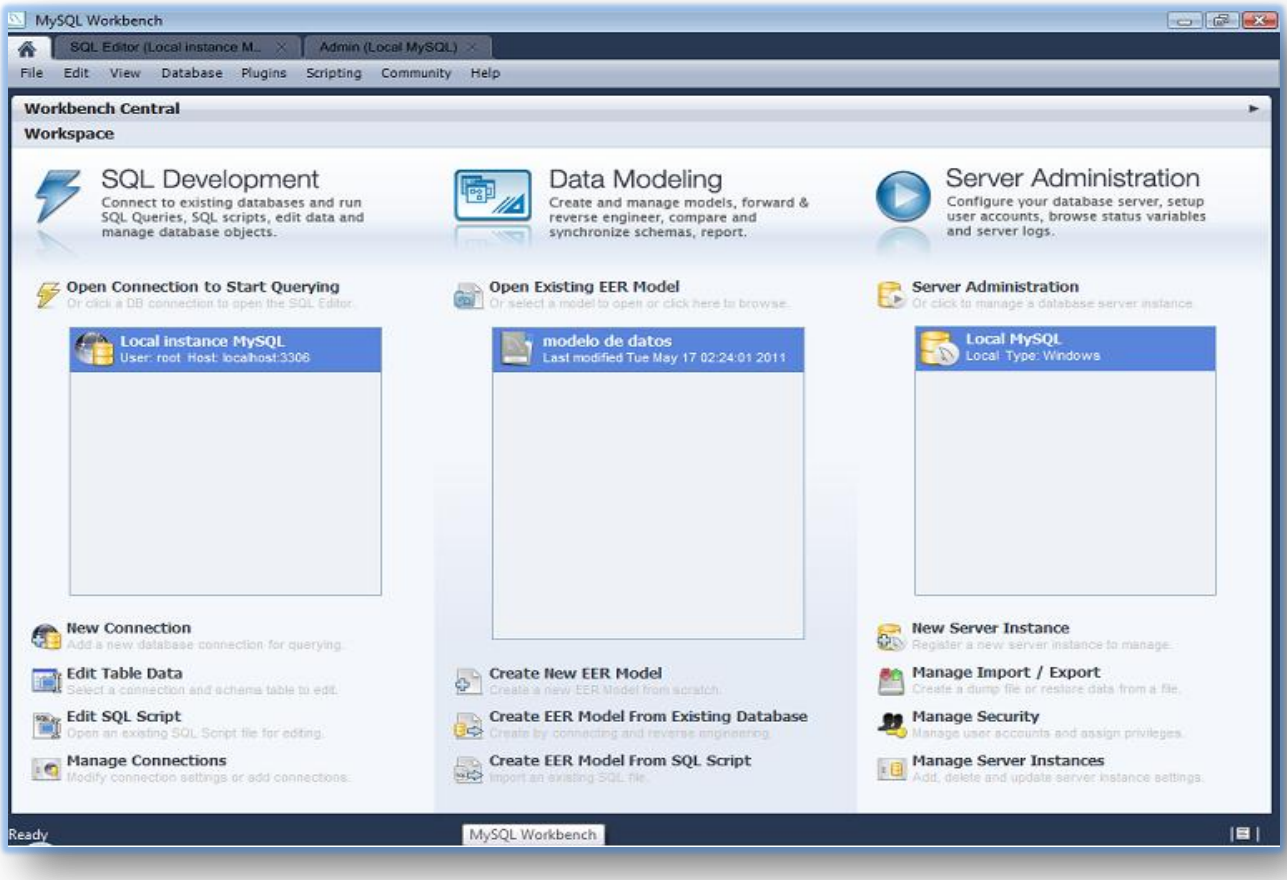
Sistema operativo Windows Vista Business: Si bien es posible utilizar otros sistemas operativos (versiones más modernas de Windows o Linux), la aplicación ha sido desarrollada y probada utilizando este sistema operativo.

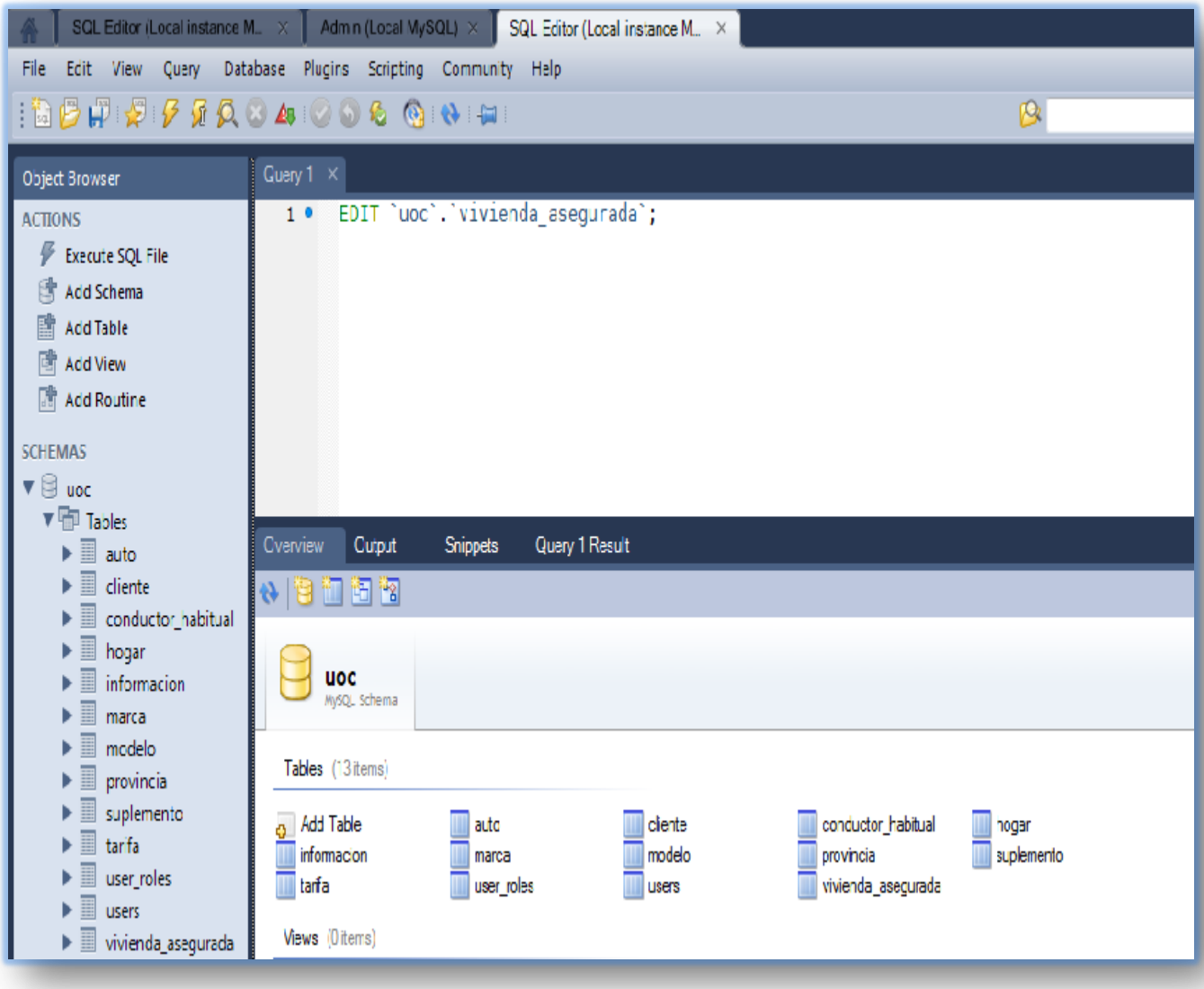
Cabe señalar que el usuario final sólo debe tener instalado en su ordenador un navegador web. Se ha probado la aplicación con los siguientes navegadores : **Internet Explorer versiones 7, 8, 9**.

Motor de base de datos MySql Community Server (http://www.mysql.com/)**:** La versión utilizada ha sido la 5.5.9. Debe realizarse la instalación estándar y configurar la contraseña del usuario root Luego es necesario ejecutar el script de creación de la base de datos denominada **uoc.**



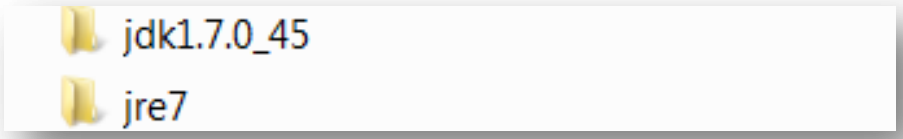
Mysql Workbench (http://wb.mysql.com/) utilizado para administrar , manipular crear Bases de datos, tablas…

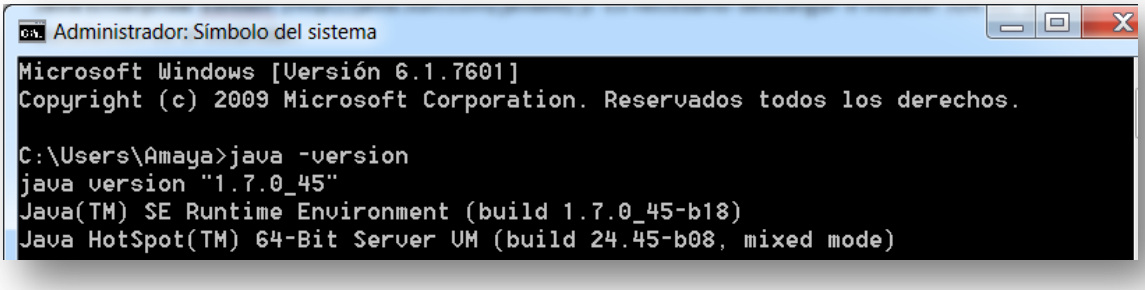




Java Enterprise Edition
(http://www.oracle.com/technetwork/java/javase/downloads/index.html):
Es necesario descargar e instalar **Java Platform (JDK) 7u45**. El proceso de instalación es el estándar.

jdk1.7.0_454
jre7

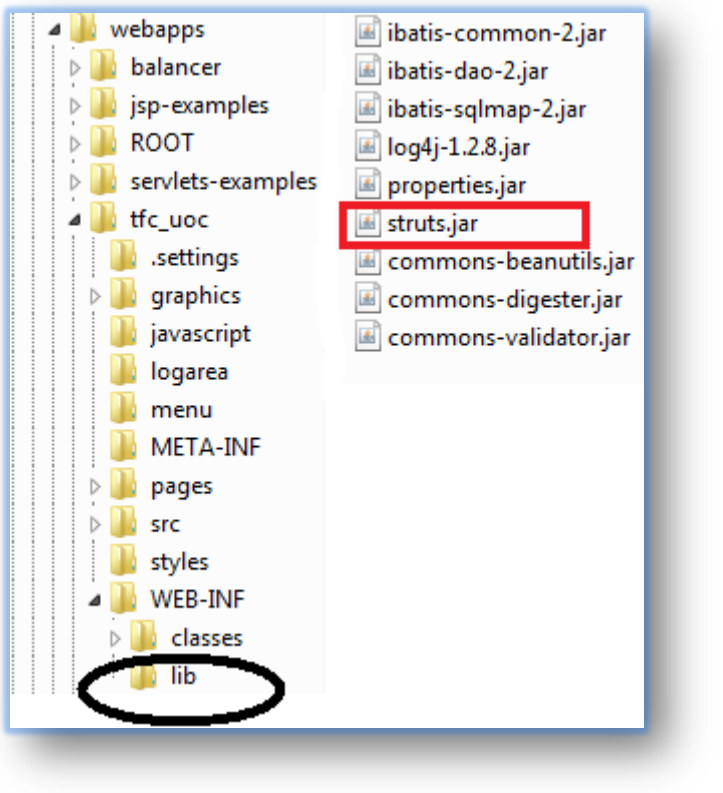




Servidor web Apache Tomcat : La versión utilizada ha sido la 5.5.33. El proceso de instalación es el estándar.

Struts Taglib: Es necesario tener esta librería para poder utilizar JavaServer Pages. Debe copiarse el fichero **struts.jar**
a la carpeta **C:\Tomcat5.5**webapps\tfc_uoc\WEB-INF\lib, donde:

C:\Tomcat5.5 es la ruta donde está instalado el Servidor Tomcat.
webapps el directorio donde se incluyen las aplicaciones webs.
tfc_uoc el directorio del proyecto.

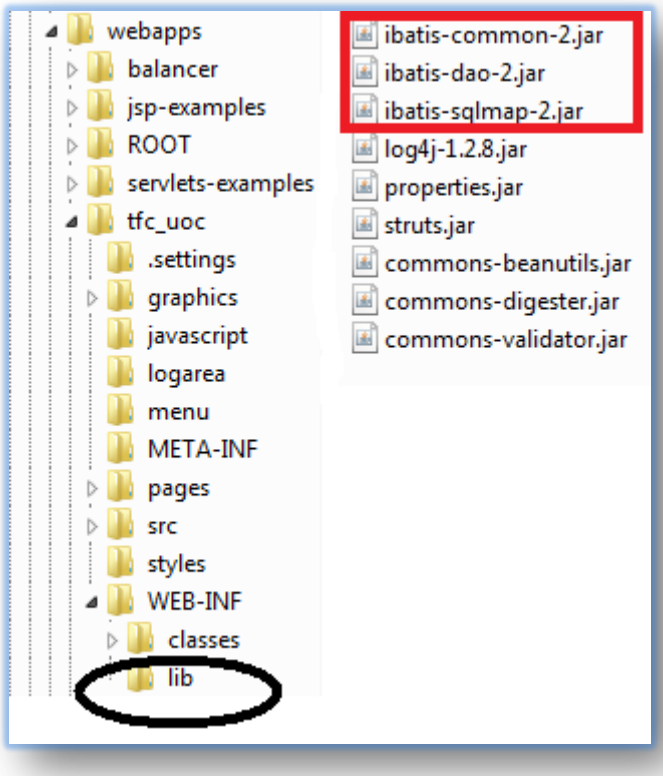


Ibatis 2 : Se utiliza para facilitar el mapeo objeto-relacional entre la base de datos y la aplicación Java.

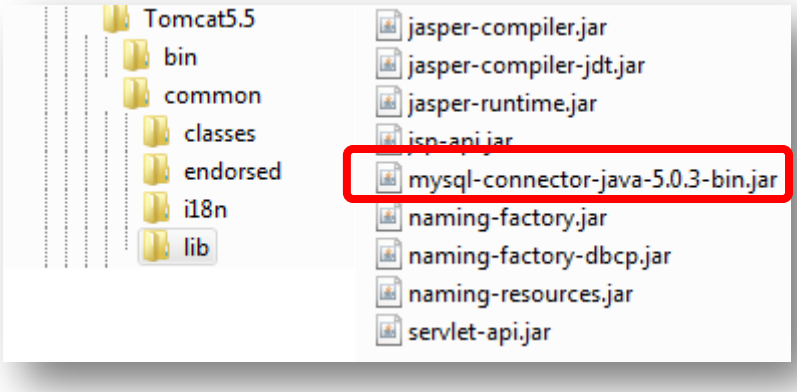
Deben copiarse los ficheros :

ibatis-common-2.jar .
ibatis-dao-2.jar .
ibatis-sqlmap-2.jar.

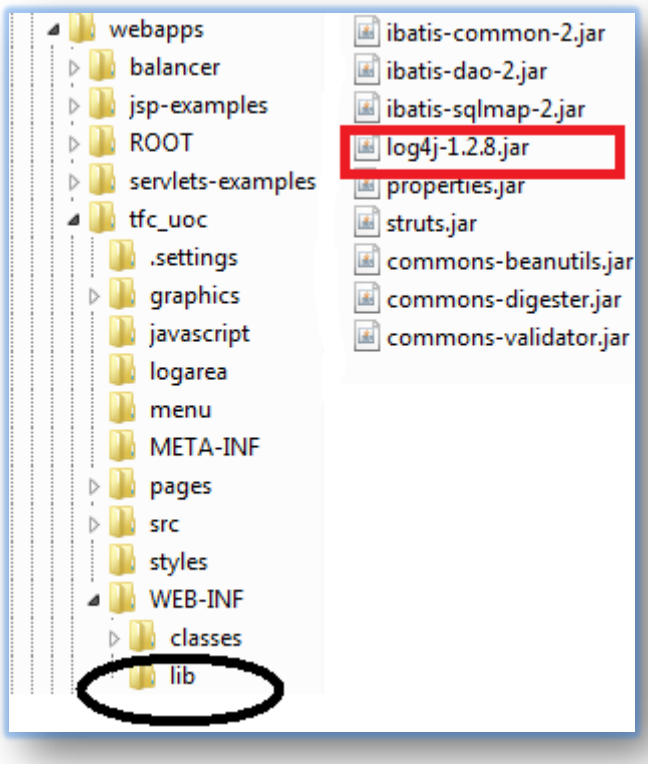
a la carpeta **C:\Tomcat5.5**webapps\tfc_uoc\WEB-INF\lib



MySql Connector Java 5.0.3-bin (http://www.mysql.com/): Lo utiliza Ibatis para conectarse a la base de datos. Este fichero debe copiarse a la carpeta **C:\Tomcat5.5\common\lib\mysql-connector-java-5.0.3-bin.jar**



Log4J 1.2.28 (http://grepcode.com/snapshot/repo1.maven.org/maven2/log4j/log4j/1.2.8): Es el componente elegido para dejar trazas de la ejecución del sistema.

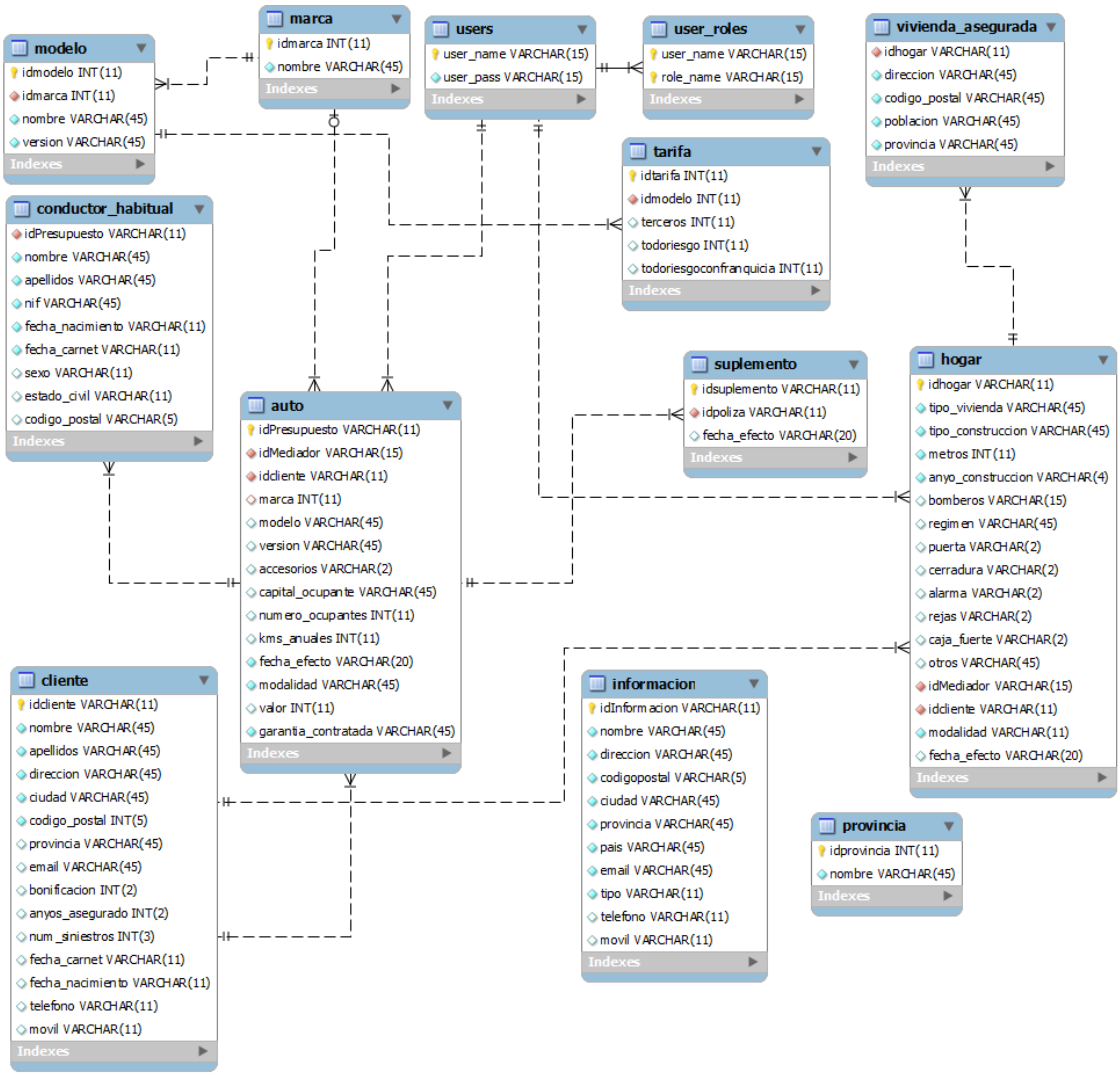


Las librerías :

commons-validator.jar (http://commons.apache.org/validator/download_validator.cgi**).**
commons-beanutils.jar(**http://commons.apache.org/beanutils/download_beanutils.cgi).**
commons-digester.jar (http://commons.apache.org/digester/download_digester.cgi):

Finalmente, cabe señalar que el usuario final sólo debe tener instalado en su ordenador un navegador web. Se ha probado la aplicación con los siguientes navegadores : **Internet Explorer 8, Internet Explorer 9.**

3.2. Script de creación de la Base de Datos




```
/* Si la BD existe la volvemos a crear. */

DROP DATABASE IF EXISTS uoc;
CREATE DATABASE uoc;

/*Creación de usuarios y roles*/

CREATE  TABLE IF NOT EXISTS `uoc`.`users` (

  `user_name` VARCHAR(15) NOT NULL ,

  `user_pass` VARCHAR(15) NOT NULL ,

  PRIMARY KEY (`user_name`) )

ENGINE = InnoDB

DEFAULT CHARACTER SET = latin1
```

```
CREATE  TABLE IF NOT EXISTS `uoc`.`user_roles` (
  `user_name` VARCHAR(15) NOT NULL ,
  `role_name` VARCHAR(15) NOT NULL ,
  PRIMARY KEY (`user_name`, `role_name`) ,
  INDEX `user_name` (`user_name` ASC) ,
  CONSTRAINT `user_name`
    FOREIGN KEY (`user_name` )
    REFERENCES `uoc`.`users` (`user_name` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1
```

```
CREATE  TABLE IF NOT EXISTS `uoc`.`marca` (
  `idmarca` INT(11) NOT NULL ,
  `nombre` VARCHAR(45) NOT NULL ,
  PRIMARY KEY (`idmarca`) )
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1
```

```
CREATE  TABLE IF NOT EXISTS `uoc`.`modelo` (
  `idmodelo` INT(11) NOT NULL ,
  `idmarca` INT(11) NOT NULL ,
  `nombre` VARCHAR(45) NOT NULL ,
  `version` VARCHAR(45) NOT NULL ,
```


```
  PRIMARY KEY (`idmodelo`) ,
  INDEX `idmarca` (`idmarca` ASC) ,
  CONSTRAINT `idmarca`
    FOREIGN KEY (`idmarca` )
    REFERENCES `uoc`.`marca` (`idmarca` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1


CREATE  TABLE IF NOT EXISTS `uoc`.`informacion` (
  `idInformacion` VARCHAR(11) NOT NULL ,
  `nombre` VARCHAR(45) NOT NULL ,
  `direccion` VARCHAR(45) NOT NULL ,
  `codigopostal` VARCHAR(5) NOT NULL ,
  `ciudad` VARCHAR(45) NOT NULL ,
  `provincia` VARCHAR(45) NOT NULL ,
  `pais` VARCHAR(45) NOT NULL ,
  `email` VARCHAR(45) NOT NULL ,
  `tipo` VARCHAR(11) NOT NULL ,
  `telefono` VARCHAR(11) NULL DEFAULT NULL ,
  `movil` VARCHAR(11) NULL DEFAULT NULL ,
  PRIMARY KEY (`idInformacion`) )
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1
CREATE  TABLE IF NOT EXISTS `uoc`.`cliente` (
  `idcliente` VARCHAR(11) NOT NULL ,
  `nombre` VARCHAR(45) NOT NULL ,
  `apellidos` VARCHAR(45) CHARACTER SET 'latin1' COLLATE 'latin1_bin' NOT NULL ,
```


```
  `direccion` VARCHAR(45) NOT NULL ,
  `ciudad` VARCHAR(45) NOT NULL ,
  `codigo_postal` INT(5) NOT NULL ,
  `provincia` VARCHAR(45) NULL DEFAULT NULL ,
  `email` VARCHAR(45) NULL DEFAULT NULL ,
  `bonificacion` INT(2) NULL DEFAULT NULL ,
  `anyos_asegurado` INT(2) NULL DEFAULT NULL ,
  `num_siniestros` INT(3) NULL DEFAULT NULL ,
  `fecha_carnet` VARCHAR(11) NULL DEFAULT NULL ,
  `fecha_nacimiento` VARCHAR(11) NULL DEFAULT NULL ,
  `telefono` VARCHAR(11) NULL DEFAULT NULL ,
  `movil` VARCHAR(11) NULL DEFAULT NULL ,
  PRIMARY KEY (`idcliente`) )
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1


CREATE  TABLE IF NOT EXISTS `uoc`.`auto` (
  `idPresupuesto` VARCHAR(11) NOT NULL ,
  `idMediador` VARCHAR(15) NOT NULL ,
  `idcliente` VARCHAR(11) NOT NULL ,
  `marca` INT(11) NULL DEFAULT NULL ,
  `modelo` VARCHAR(45) NULL DEFAULT NULL ,
  `version` VARCHAR(45) NULL DEFAULT NULL ,
  `accesorios` VARCHAR(2) NULL DEFAULT NULL ,
  `capital_ocupante` VARCHAR(45) NULL DEFAULT NULL ,
  `numero_ocupantes` INT(11) NULL DEFAULT NULL ,
  `kms_anuales` INT(11) NULL DEFAULT NULL ,
  `fecha_efecto` VARCHAR(20) NOT NULL ,
  `modalidad` VARCHAR(45) NOT NULL ,
```


```
  `valor` INT(11) NULL DEFAULT NULL ,
  `garantia_contratada` VARCHAR(45) NOT NULL ,
  PRIMARY KEY (`idPresupuesto`) ,
  INDEX `marca` (`marca` ASC) ,
  INDEX `idMediador` (`idMediador` ASC) ,
  INDEX `idcliente` (`idcliente` ASC) ,
  CONSTRAINT `idcliente`
    FOREIGN KEY (`idcliente` )
    REFERENCES `uoc`.`cliente` (`idcliente` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `idMediador`
    FOREIGN KEY (`idMediador` )
    REFERENCES `uoc`.`users` (`user_name` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `marca`
    FOREIGN KEY (`marca` )
    REFERENCES `uoc`.`marca` (`idmarca` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1


CREATE  TABLE IF NOT EXISTS `uoc`.`conductor_habitual` (
  `idPresupuesto` VARCHAR(11) NOT NULL ,
  `nombre` VARCHAR(45) NOT NULL ,
  `apellidos` VARCHAR(45) NOT NULL ,
  `nif` VARCHAR(45) NOT NULL ,
```


```
  `fecha_nacimiento` VARCHAR(11) NOT NULL ,
  `fecha_carnet` VARCHAR(11) NOT NULL ,
  `sexo` VARCHAR(11) NULL DEFAULT NULL ,
  `estado_civil` VARCHAR(11) NULL DEFAULT NULL ,
  `codigo_postal` VARCHAR(5) NULL DEFAULT NULL ,
  INDEX `idPresupuesto` (`idPresupuesto` ASC) ,
  CONSTRAINT `idPresupuesto`
    FOREIGN KEY (`idPresupuesto` )
    REFERENCES `uoc`.`auto` (`idPresupuesto` )
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1


CREATE  TABLE IF NOT EXISTS `uoc`.`hogar` (
  `idhogar` VARCHAR(11) NOT NULL ,
  `tipo_vivienda` VARCHAR(45) NOT NULL ,
  `tipo_construccion` VARCHAR(45) NOT NULL ,
  `metros` INT(11) NOT NULL ,
  `anyo_construccion` VARCHAR(4) NOT NULL ,
  `bomberos` VARCHAR(15) NULL DEFAULT NULL ,
  `regimen` VARCHAR(45) NULL DEFAULT NULL ,
  `puerta` VARCHAR(2) NULL DEFAULT NULL ,
  `cerradura` VARCHAR(2) NULL DEFAULT NULL ,
  `alarma` VARCHAR(2) NULL DEFAULT NULL ,
  `rejas` VARCHAR(2) NULL DEFAULT NULL ,
  `caja_fuerte` VARCHAR(2) NULL DEFAULT NULL ,
  `otros` VARCHAR(45) NULL DEFAULT NULL ,
  `idMediador` VARCHAR(15) NOT NULL ,
```


```
  `idcliente` VARCHAR(11) NOT NULL ,
  `modalidad` VARCHAR(11) NOT NULL ,
  `fecha_efecto` VARCHAR(20) NULL DEFAULT NULL ,
  PRIMARY KEY (`idhogar`) ,
  INDEX `idMediador` (`idMediador` ASC) ,
  INDEX `idcliente` (`idcliente` ASC) ,
  CONSTRAINT `idclient`
    FOREIGN KEY (`idcliente` )
    REFERENCES `uoc`.`cliente` (`idcliente` )
    ON DELETE CASCADE,
  CONSTRAINT `idMediado`
    FOREIGN KEY (`idMediador` )
    REFERENCES `uoc`.`users` (`user_name` )
    ON DELETE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1

CREATE  TABLE IF NOT EXISTS `uoc`.`vivienda_asegurada` (
  `idhogar` VARCHAR(11) NOT NULL ,
  `direccion` VARCHAR(45) NOT NULL ,
  `codigo_postal` VARCHAR(45) NOT NULL ,
  `poblacion` VARCHAR(45) NOT NULL ,
  `provincia` VARCHAR(45) NOT NULL ,
  INDEX `idhogar` (`idhogar` ASC) ,
  CONSTRAINT `idhogar`
    FOREIGN KEY (`idhogar` )
    REFERENCES `uoc`.`hogar` (`idhogar` )
    ON DELETE CASCADE
    ON UPDATE CASCADE)
```


```
ENGINE = InnoDB

DEFAULT CHARACTER SET = latin1

CREATE  TABLE IF NOT EXISTS `uoc`.`suplemento` (

 `idsuplemento` VARCHAR(11) NOT NULL ,

 `idpoliza` VARCHAR(11) NOT NULL ,

 `fecha_efecto` VARCHAR(20) NULL DEFAULT NULL ,

 PRIMARY KEY (`idsuplemento`) ,

 INDEX `idpoliza` (`idpoliza` ASC) ,

 CONSTRAINT `idPoliza`

  FOREIGN KEY (`idpoliza` )

  REFERENCES `uoc`.`auto` (`idPresupuesto` )

  ON DELETE CASCADE

  ON UPDATE CASCADE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = latin1
```

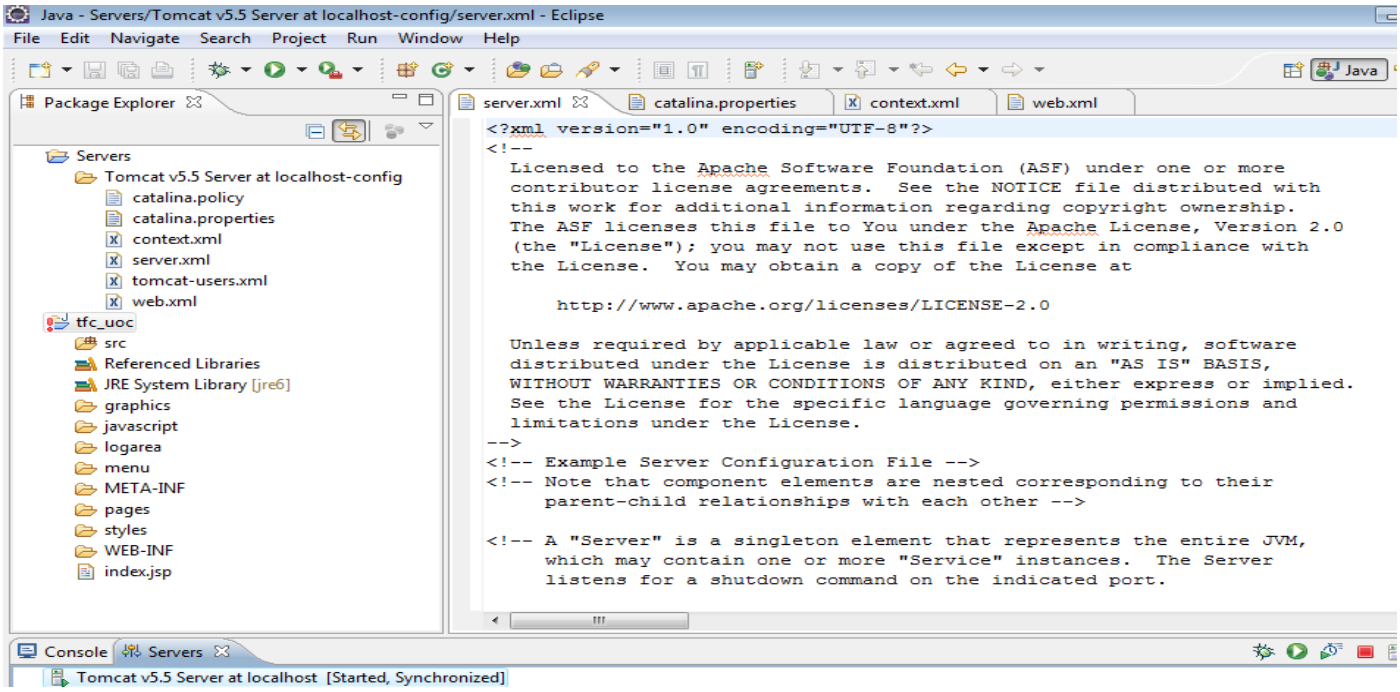
En la tabla marca se insertan las diferentes marcas de vehículos, evidentemente no están todas reflejadas sólo una muestra significativa.

```
INSERT INTO `uoc`.`marca` (`idmarca`, `nombre`) VALUES (1, 'AUDI');
INSERT INTO `uoc`.`marca` (`idmarca`, `nombre`) VALUES (2, 'AUSTIN');
INSERT INTO `uoc`.`marca` (`idmarca`, `nombre`) VALUES (3, 'BMW');
INSERT INTO `uoc`.`marca` (`idmarca`, `nombre`) VALUES (4, 'CHEVROLET');
INSERT INTO `uoc`.`marca` (`idmarca`, `nombre`) VALUES (5, 'FIAT');
INSERT INTO `uoc`.`marca` (`idmarca`, `nombre`) VALUES (6, 'SEAT');
INSERT INTO `uoc`.`marca` (`idmarca`, `nombre`) VALUES (7, 'RENAULT');
INSERT INTO `uoc`.`marca` (`idmarca`, `nombre`) VALUES (8, 'MERCEDES');
INSERT INTO `uoc`.`marca` (`idmarca`, `nombre`) VALUES (9, 'OPEL');
INSERT INTO `uoc`.`marca` (`idmarca`, `nombre`) VALUES (10, 'HONDA');
INSERT INTO `uoc`.`marca` (`idmarca`, `nombre`) VALUES (11, 'TOYOTA');
INSERT INTO `uoc`.`marca` (`idmarca`, `nombre`) VALUES (12, 'MITSUBISHI');
INSERT INTO `uoc`.`marca` (`idmarca`, `nombre`) VALUES (13, 'VOLKSWAGEN');
INSERT INTO `uoc`.`marca` (`idmarca`, `nombre`) VALUES (14, 'VOLVO');
INSERT INTO `uoc`.`marca` (`idmarca`, `nombre`) VALUES (15, 'TATA');
INSERT INTO `uoc`.`marca` (`idmarca`, `nombre`) VALUES (16, 'HYUNDAI');
INSERT INTO `uoc`.`marca` (`idmarca`, `nombre`) VALUES (17, 'CITROEN');
INSERT INTO `uoc`.`marca` (`idmarca`, `nombre`) VALUES (18, 'PEUGEOT');
```

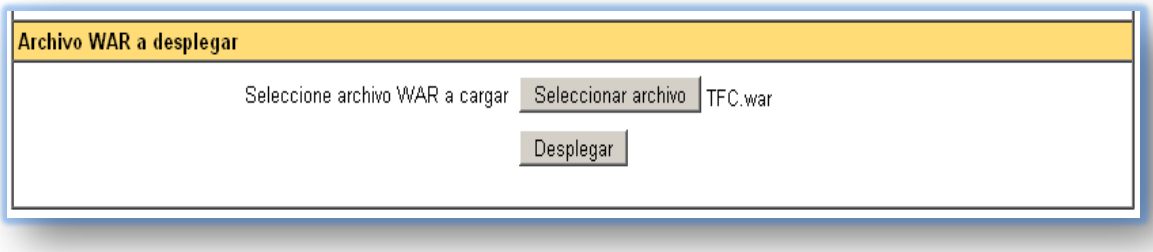

```
INSERT INTO `uoc`.`marca` (`idmarca`, `nombre`) VALUES (19, 'SAAB');
INSERT INTO `uoc`.`marca` (`idmarca`, `nombre`) VALUES (20, 'MAZDA');
```

3.3. Instalación de la aplicación

Para hacer posible la implementación de este proyecto y como se estableció al principio se ha utilizado Eclipse para escribir todo el código de la aplicación, compilar éste y depurarlo. Además al establecerse como servidor Tomcat , se arranca desde Eclipse ,de tal manera que es automática la tarea de ver los cambios de código y su ejecución en un navegador web.

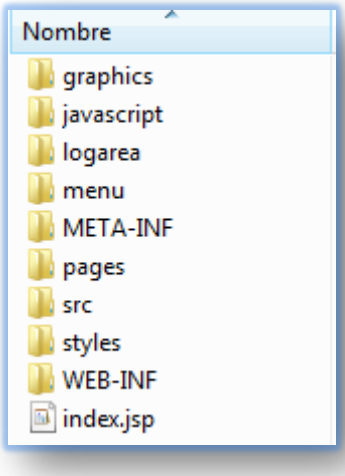


Una vez instalados el servidor web y la base de datos estamos en condiciones de instalar la aplicación Gestión de Seguros. La misma se encuentra empaquetada en un fichero WAR (**tfc_uoc.war**). Debemos ir al Tomcat Manager y seleccionar desplegar **tfc_uoc.war**



Desplegar WAR

Esta acción despliega las siguientes carpetas en el directorio ${catalina.home}\webapps\tfc_uoc:



graphics : Contiene las imágenes por utilizadas en las diferentes páginas de la aplicación web.

javascript : Contiene todos los ficheros javascript que son utilizados en las diferentes páginas JSP's.

logarea : Contiene los ficheros que hacen posible devolver información de lo ocurrido en la aplicación en el área de mensajes en todas las páginas JSP's.

menu : Contiene la página JSP de la aplicación que construye el menú.

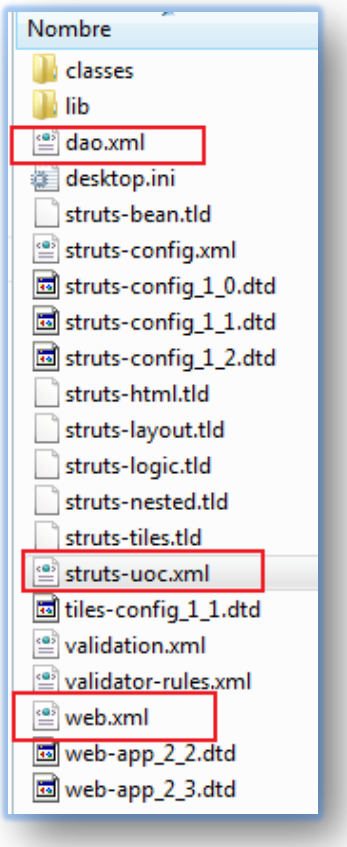
META-INF: Contiene el fichero manifest.

Pages : Contiene las páginas JSP's de la aplicación.

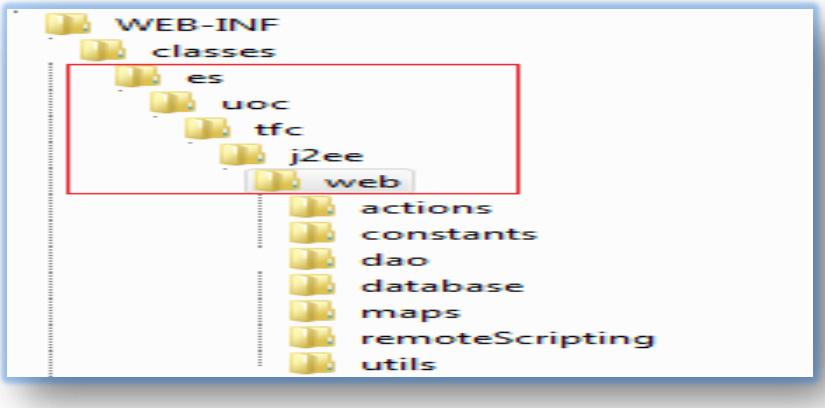
src: Contiene el código fuente de java.

styles : Contiene el fichero de estilos **estilos.css**

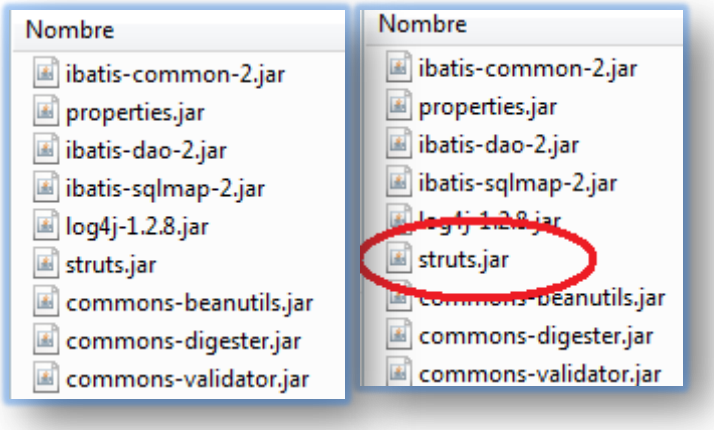
WEB-INF: contiene los ficheros de configuración web.xml, dao.xml, struts-uoc.xml… entre otros.



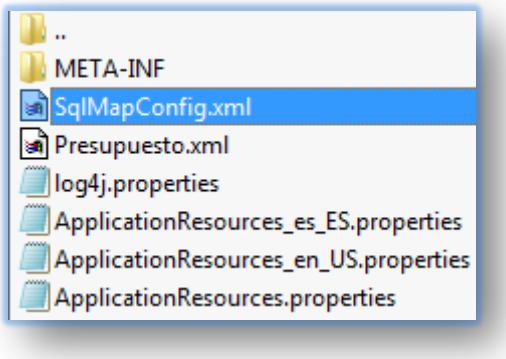
WEB-INF/classes: contiene los packages con las clases y fuentes de la aplicación.



WEB-INF/lib: contiene las librerías necesarias para el funcionamiento del sistema (Ibatis, Log4J, etc.)



Un aspecto a reseñar sobre la librería **properties.jar** , importante puesto que contiene ficheros esenciales tales como **SqlMapConfig.xml, Presupuesto.xml(contiene todas las sql de la aplicación),log4j.properties** que configura donde se crearán y cómo los ficheros logs de la aplicación.



Por ejemplo éste sería la configuración del fichero log4j.properties contenido en el fichero properties.jar

```
#############################################################################
# Log4Java configuration file for Fundacion
#############################################################################

#Global configuration
```

```
log4j.appender.UOCLog=org.apache.log4j.DailyRollingFileAppender
log4j.appender.UOCLog.File=C:/Tomcat5.5/logs/UOC.log
log4j.appender.UOCLog.DatePattern='.'yyyy-MM-dd
log4j.appender.UOCLog.layout=org.apache.log4j.PatternLayout
log4j.appender.UOCLog.layout.ConversionPattern=%d %-5p - %m%n

#database
log4j.logger.es.uoc.tfc.j2ee.web.database.SqlMapConfig=DEBUG, UOCLog
log4j.logger.es.uoc.tfc.j2ee.web.database.DataAccess=DEBUG, UOCLog
log4j.logger.es.uoc.tfc.j2ee.web.database.DBConnection=DEBUG, UOCLog

#utils
log4j.logger.es.uoc.tfc.j2ee.web.utils.Comun=DEBUG, UOCLog
log4j.logger.es.uoc.tfc.j2ee.web.utils.PropertiesManager=DEBUG, UOCLog

#beans
log4j.logger.es.uoc.tfc.j2ee.web.beans.User=DEBUG, UOCLog
log4j.logger.es.uoc.tfc.j2ee.web.beans.DetalleTipoHorarioBean=DEBUG, UOCLog
log4j.logger.es.uoc.tfc.j2ee.web.beans.UserBean=DEBUG, UOCLog

#iBatis
log4j.logger.com.ibatis=DEBUG, UOCLog
log4j.logger.com.ibatis.common.jdbc.SimpleDataSource=DEBUG, UOCLog
log4j.logger.com.ibatis.sqlmap.engine.impl.SqlMapClientDelegate=DEBUG, UOCLog
log4j.logger.java.sql.Statement=DEBUG, UOCLog

#actions
log4j.logger.es.uoc.tfc.j2ee.web.actions.PresupuestoAutoAction=INFO, UOCLog
log4j.logger.es.uoc.tfc.j2ee.web.actions.PresupuestoHogarAction=INFO, UOCLog
log4j.logger.es.uoc.tfc.j2ee.web.actions.ClienteAction=INFO, UOCLog
```

Antes de comenzar a ejecutar la aplicación es necesario realizar algunos ajustes en la configuración.

1. En el fichero Server.xml hay que especificar un tema muy importante puesto que de no hacerse no podríamos conectarnos a la aplicación . Teniendo en cuenta que he elegido para la conexión Jdbcrealm que es una implementación de la interfaz de Tomcat que busca los usuarios en una base de datos relacional accediendo a través de un driver JDBC ,es necesario haber creado las tablas users (user_name,user_pass) ,roles(user_name,role_name) . Una vez creadas estas tablas configuraremos en el Server xml lo siguiente:

```
<Realm  className="org.apache.catalina.realm.JDBCRealm"

      driverName="org.gjt.mm.mysql.Driver"
   connectionURL="jdbc:mysql://localhost/uoc"
  connectionName="root" connectionPassword="root"
     userTable="users" userNameCol="user_name" userCredCol="user_pass"
  userRoleTable="user_roles" roleNameCol="role_name" />
```

Además para que funcione la aplicación sin problemas añadimos estas líneas en el mismo fichero :

```
<Context path="/tfc-uoc" docBase="C:\Tomcat5.5\webapps\tfc_uoc" debug="5" reloadable="true">
<Resource name="jdbc/database" auth="Container" type="javax.sql.DataSource" maxActive="100"
maxIdle="30" maxWait="10000" username="root" password="root" driverClassName="com.mysql.jdbc.Driver"
url="jdbc:mysql://localhost/uoc?autoReconnect=true"/>
```

```
</Context>
```

2. En el fichero web.xml de la aplicación debemos configurar unas cuantas:

```
<!-- Adiciones de configuracion de Struts -->
<!-- Standard Action Servlet Configuration (with debugging) -->
<servlet>
  <servlet-name>action</servlet-name>
  <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>

  <init-param>
    <param-name>config</param-name>
    <param-value>/WEB-INF/struts-config.xml,/WEB-INF/struts-uoc.xml</param-value>
  </init-param>
  <init-param>
    <param-name>debug</param-name>
    <param-value>2</param-value>
  </init-param>
  <init-param>
    <param-name>detail</param-name>
    <param-value>2</param-value>
  </init-param>
  <load-on-startup>2</load-on-startup>
</servlet>
```

```
<!-- Standard Action Servlet Mapping -->
<servlet-mapping>
  <servlet-name>action</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>

  <!-- Fin de Adiciones de configuracion de Struts -->
```

Para poder entrar con los permisos correctos teniendo en cuenta las restricciones que anteriormente he expuesto en la conexión Jdbcrealm, debemos incluir esto en el fichero web.xml:

```
<login-config>
   <auth-method>FORM</auth-method>
   <realm-name>Form-Based Authentication</realm-name>
   <form-login-config>
     <form-login-page>/pages/auth/WEBIndex.jsp</form-login-page>
     <form-error-page>/pages/auth/loginError.jsp</form-error-page>
   </form-login-config>
 </login-config>


          <!-- Security roles referenced by this web application -->

<security-role>
     <role-name>produccion</role-name>
     <role-name>colaborador</role-name>
   </security-role>

  <security-constraint>
```


```
        <web-resource-collection>
                <web-resource-name>Paginas</web-resource-name>
                <description> accessible by authorised users </description>
                <url-pattern>/pages/*</url-pattern>
        <url-pattern>/menu/dynamicMenu.jsp</url-pattern>
        <http-method>GET</http-method>
                <http-method>POST</http-method>
        </web-resource-collection>
  <auth-constraint>
        <description>These are the roles who have access</description>

   <role-name>produccion</role-name>
          <role-name>colaborador</role-name>
        </auth-constraint>
 </security-constraint>
```

Datasource también debe incluirse en el fichero web.xml:

```
<resource-ref>
<description>BD AppDB</description>
<res-ref-name>jdbc/database</res-ref-name>
<res-type>javax.sql.DataSource</res-type>
<res-auth>Container</res-auth>
</resource-ref>
```

Para establecer las páginas de error :

```
<!-- HTTP error pages >
 <error-page>
        <error-code>404</error-code>
                <location>/pages/masterFiles/error/error.jsp</location>
        </error-page>
 <error-page>
                <error-code>400</error-code>
                <location>/pages/masterFiles/error/error.jsp</location>
 </error-page>-->
 <error-page>
                <error-code>403</error-code>
                <location>/pages/auth/loginError.jsp</location>
 </error-page>
```

Si quisiéramos traducir nuestra aplicación a otros idiomas, es necesario utilizar estas tabligs que nos proporciona apache y que debemos incluir de este modo :

```
<!-- Internacionalizacion con taglibs -->

 <!-- <taglib>
   <taglib-uri>http://jakarta.apache.org/taglibs/i18n-1.0</taglib-uri>
   <taglib-location> /WEB-INF/taglibs-i18n.tld </taglib-location>
```


```
 </taglib> -->

<!-- Fin Internacionalizacion -->
```

Para poder utilizar las taglibs de struts en los JSP's:

```
<!-- Struts -->

  <!-- Struts Tag Library Descriptors -->
<taglib>
  <taglib-uri>/tags/struts-bean</taglib-uri>
  <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
 </taglib>

 <taglib>
  <taglib-uri>/tags/struts-html</taglib-uri>
  <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
 </taglib>

 <taglib>
  <taglib-uri>/tags/struts-logic</taglib-uri>
  <taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
 </taglib>

 <taglib>
  <taglib-uri>/tags/struts-nested</taglib-uri>
  <taglib-location>/WEB-INF/struts-nested.tld</taglib-location>
 </taglib>

 <taglib>
  <taglib-uri>/tags/struts-tiles</taglib-uri>
  <taglib-location>/WEB-INF/struts-tiles.tld</taglib-location>
 </taglib>

<!-- Fin Struts -->
```

3. En el fichero log4j.properties es posible cambiar donde se dejarán los logs:

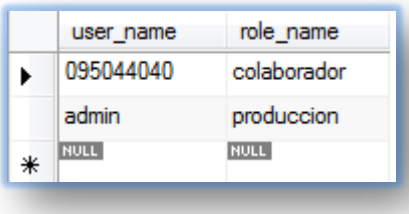
(Explicado anteriormente cuando se alude a la librería **properties.jar**)

4. Hay que dar de alta al menos un usuario Colaborador y de Produccion:

Tabla **users:**

	user_name	user_pass
▶	095044040	uoc
	admin	admin

Tabla **roles**:



user_name	role_name
095044040	colaborador
admin	produccion
NULL	NULL

INSERT INTO uoc.users (user_name,user_pass) VALUES ('095044040',' uoc');
INSERT INTO uoc.roles (user_name,role_name) VALUES ('095044040',' colaborador');

INSERT INTO uoc.users (user_name,user_pass) VALUES ('admin','admin');
INSERT INTO uoc.roles (user_name,role_name) VALUES ('admin','produccion');

Bibliografía

Booch, Grady; Rumbaugh, James; Jacobson, Ivar (1999). El lenguaje unificado de modelado. Addison Wesley.

Connolly, Thomas; Begg, Carolyn (2005). Database Systems: A Practical Approach to Design, Implementation, and Management (4ª edición). Addison Wesley.

Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John (1995). Patrones de Diseño. Addison Wesley.

Geary, David (2002, 29 de Noviembre). "A first look at JavaServer Faces, Part 1". Javaworld. http://www.javaworld.com/javaworld/jw-11-2002/jw-1129-jsf.html.

Geary, David (2002, 27 de Diciembre). "A first look at JavaServer Faces, Part 2". Javaworld. http://www.javaworld.com/javaworld/jw-12-2002/jw-1227-jsf2.html.

Larman, Craig (2005). Applying UML and patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3ª edición). Prentice Hall.

Mahmoud, Qusay H. (2004, Agosto). "Developing Web Applications with JavaServer Faces". Sun Developer Network. http://java.sun.com/developer/technicalArticles/GUI/JavaServerFaces/.

Murach, Joel; Steelman, Andrea (2008). Java Servlets and JSP (2ª edición). Murach.

Prakash Joshi, Anand (2005, 2 de Diciembre). "Design with the JSF architecture". IBM DeveloperWorks. http://www.ibm.com/developerworks/java/library/wa-dsgnpatjsf.html?ca=drs-.

Pressman, Roger S. (2006). Ingeniería del software: Un enfoque práctico (6ª edición). McGraw-Hill.

Roman, Ed; Patel Sriganesh, Rima; Brose, Gerald (2005). Mastering Enterprise JavaBeans (3ª edición). Wiley.

Anexos

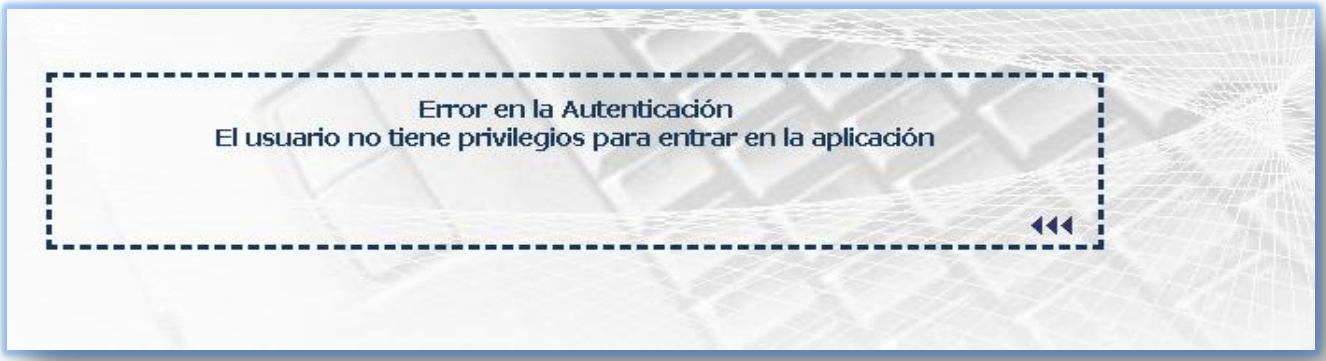
[Imágenes del sitio web]

A continuación se muestran algunas capturas de pantalla del sitio web desarrollado.

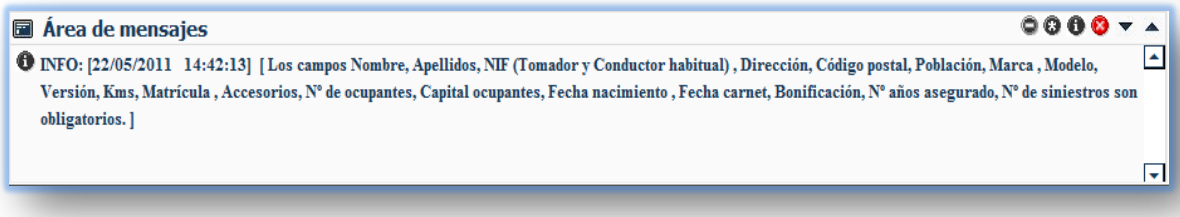
Pantalla Login

Para entrar a la aplicación lo hacemos a través de una pantalla de acceso. Para poder acceder al sistema, el usuario y password deben ser correctos y pertenecer al rol correspondiente .Teniendo en cuenta que tenemos dos perfiles **Colaborador y Jefe de Producción** , limitara el acceso al usuario dependiendo de las funcionalidades a las que tenga permisos.

Si el usuario o el password no son correctos la aplicación nos muestra una pantalla de error .



Área de Mensajes



Borrar los mensajes.

Muestra todos los mensajes.

Muestra sólo los mensajes de información.

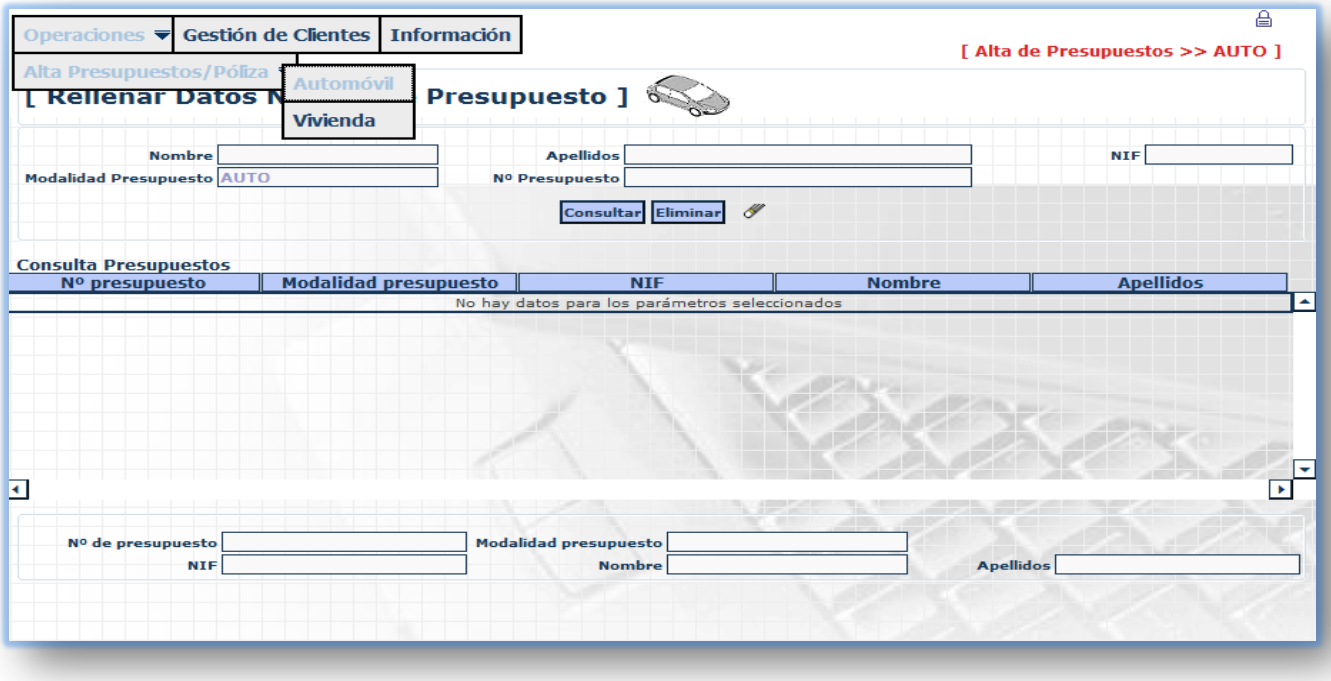
Muestra sólo los mensajes de error.

Minimizar y maximizar respectivamente.

Operaciones

Alta de presupuestos

El sistema permitirá la administración de la información necesaria para la gestión de los presupuestos de un automóvil. Para acceder es necesario tener el rol de colaborador como se ha dicho anteriormente.

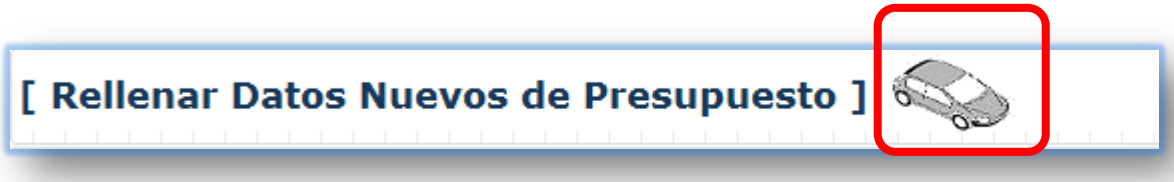


Pantalla para gestionar presupuestos de Auto.

El sistema permitirá el alta, baja (física), y consulta de presupuestos en la Base de Datos.

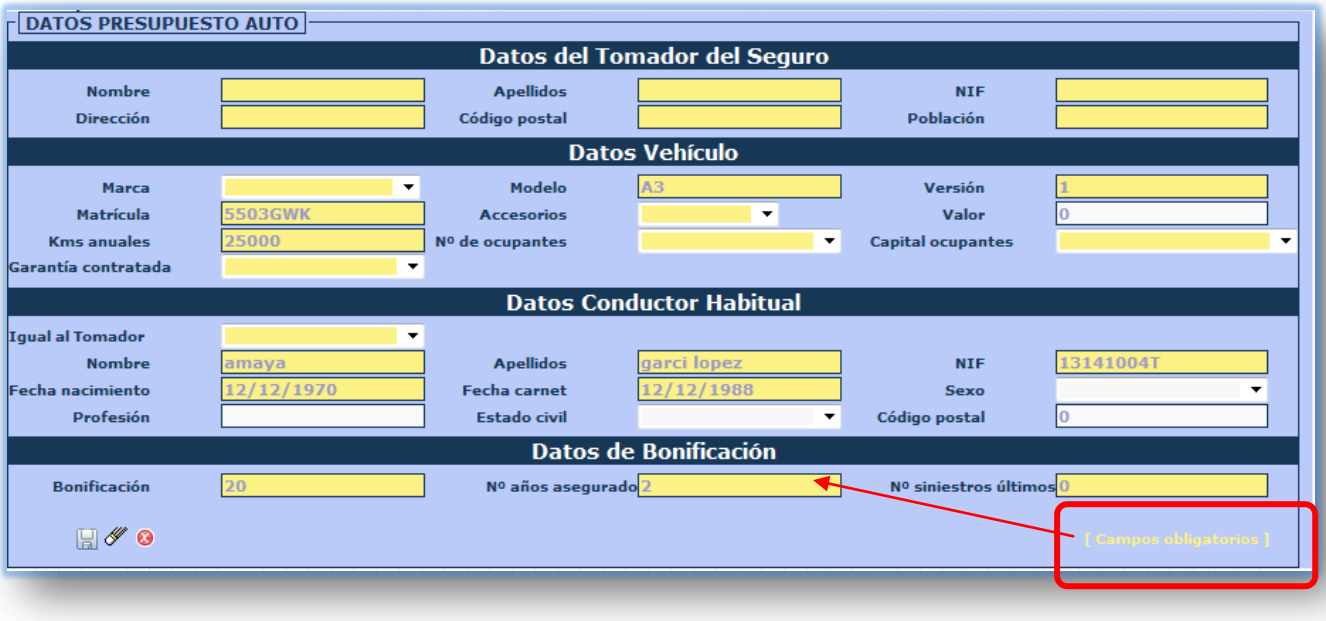
Alta

Al pulsar sobre el icono del coche



Se mostrará una pantalla desde la que se podrá dar de alta un nuevo presupuesto de auto.

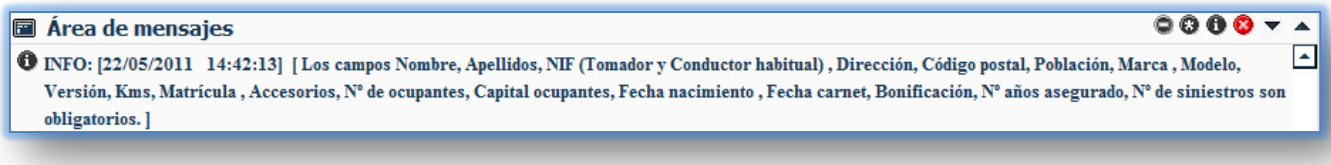
En la parte central se abrirá un formulario solicitando la siguiente información (los campos obligatorios para el usuario se identificarán con el color amarillo):



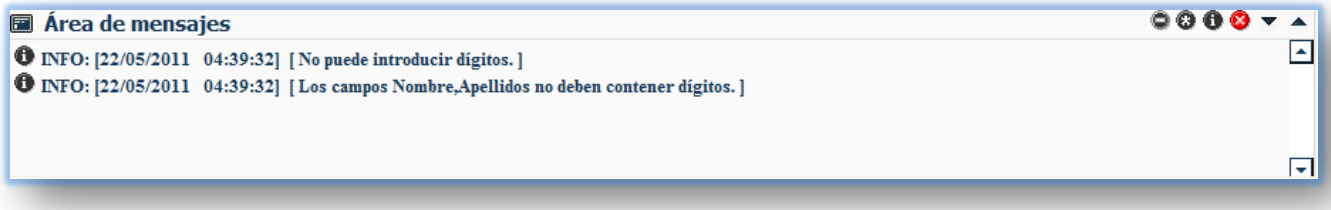
Y contendrá los siguientes botones:

Disquete  al hacer clic y antes de guardar la información del formulario para el alta de un nuevo presupuesto de un auto ,validará los campos del formulario, teniendo en cuenta los campos obligatorios y validaciones del NIF, fechas…,una vez realizadas éstas se procederá a insertar en Base de datos el contenido de éste.

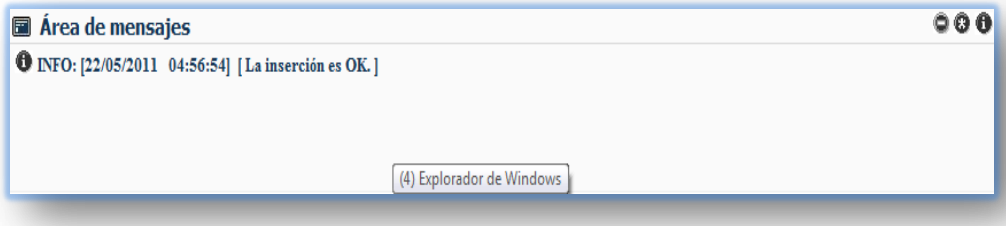
Si se pulsa sobre este icono sin haber introducido todos los datos se mostrará un aviso en el área de mensajes de la página indicándolo, puesto que éstos son campos obligatorios en la inserción.



Si los campos obligatorios introducidos no son correctos la aplicación nos advierte en el área de mensajes, que los campos no son los correctos, es decir se valida que cada campo sea el correcto.



Si la inserción en Base de Datos ha sido satisfactoria, se mostrará por pantalla dicha información en la **Consulta Presupuestos**, permaneciendo además los datos en la parte del frame superior.



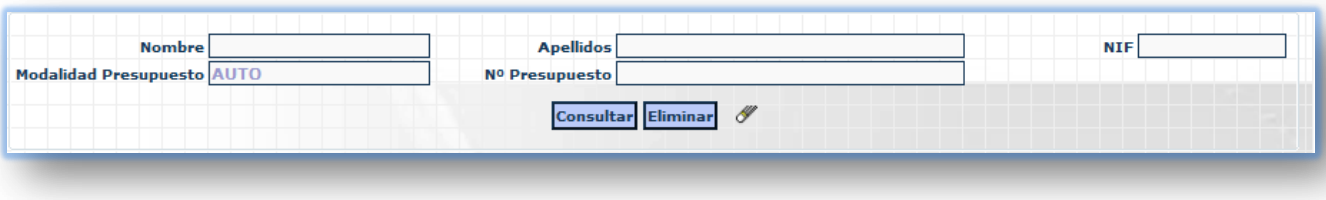
Este procedimiento inserta en la Tabla uoc.Auto (Datos Vehículo), Tabla uoc.conductor_habitual(Datos Conductor_Habitual) y uoc.cliente (Datos del Tomador del Seguro), pero esto sólo si el cliente no existiera .

Goma de borrar  al hacer clic borra los campos del formulario.

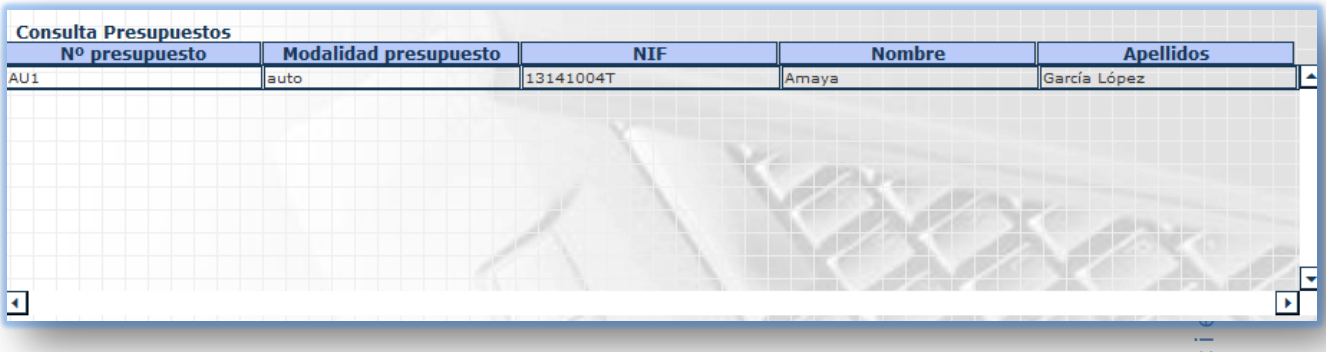
Este icono  al hacer clic cierra el formulario Datos Presupuesto Auto.

Consulta

Desde la pantalla de alta se podrá realizar una consulta introduciendo cualquiera de los campos que se muestran y pulsando sobre el botón  .



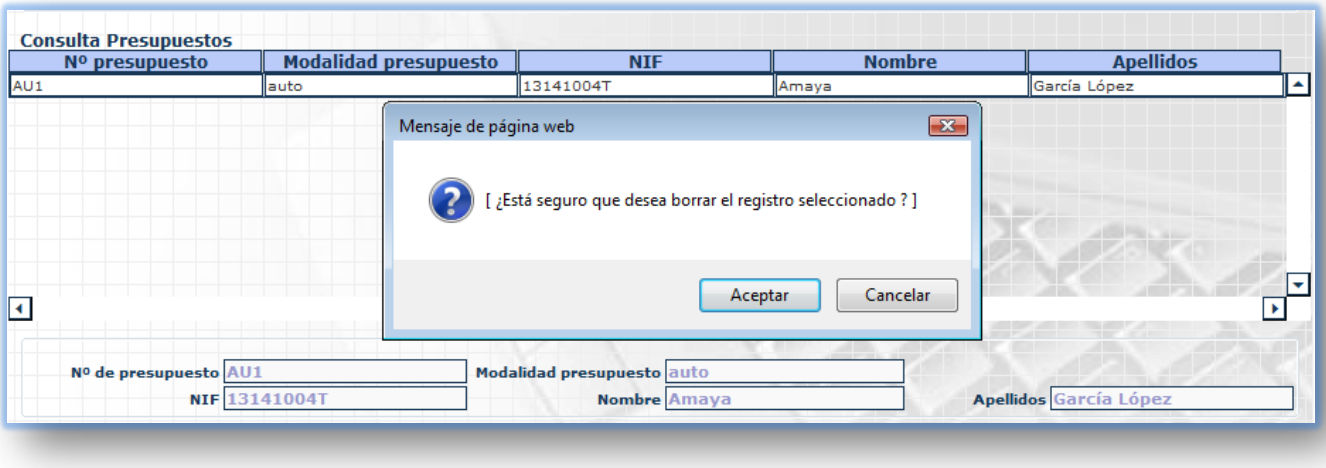
Como resultado, y tras a acceder a la tablas **uoc.cliente y uoc.auto** con los datos introducidos para obtener la información, se mostrará en la parte inferior denominada **Consulta Presupuestos** un listado con la consulta obtenida.



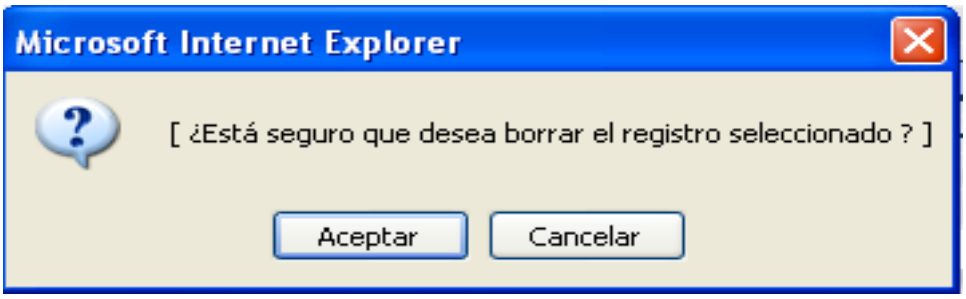
Baja

Para dar de baja el registro deseado pulsamos sobre el botón correspondiente 

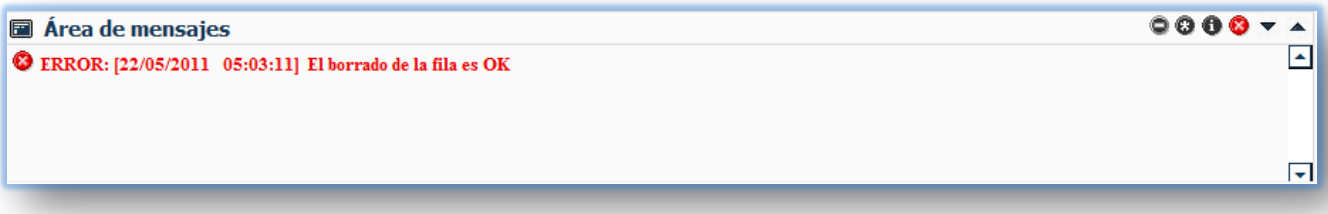
, pero antes haber seleccionado el registro que queremos borrar y cuyos datos aparecen en el frame inferior.



Antes de proceder al borrado del registro en Base de Datos, se pedirá confirmación al usuario.



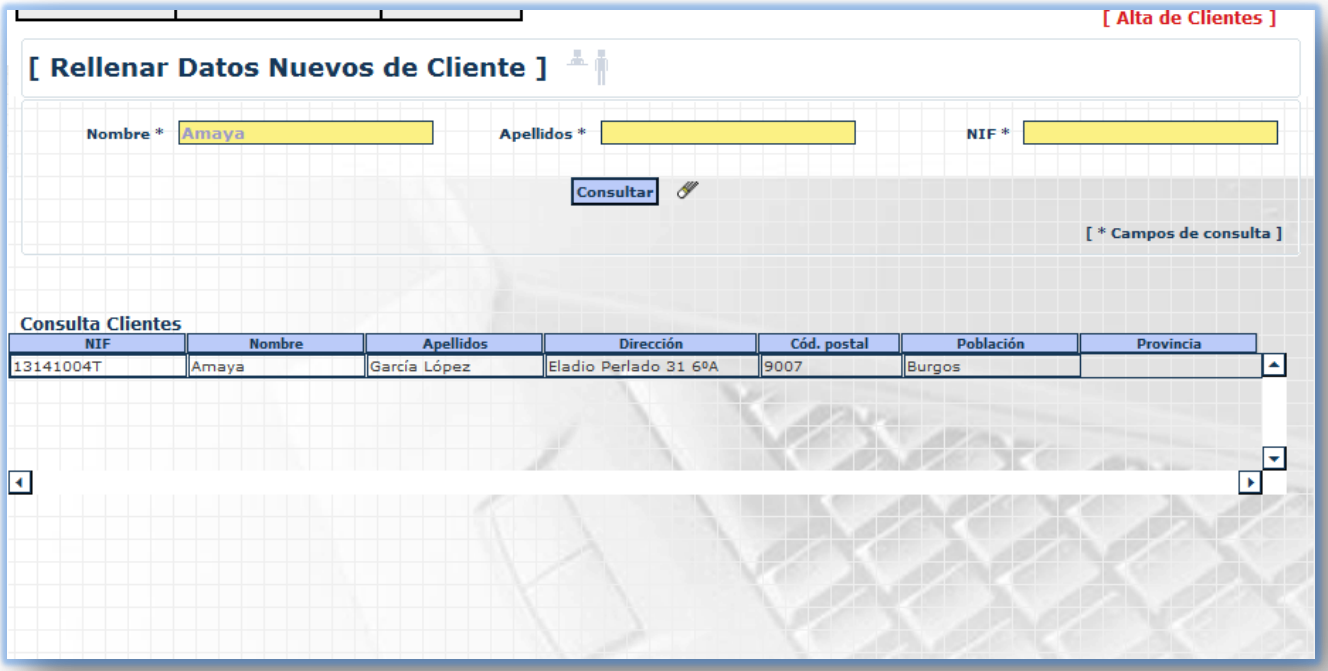
Si el borrado es correcto se mostrará dicho mensaje



Si el borrado no fuera correcto se mostrará el mensaje de error pertinente.

Una vez realizada la baja, se mostrará el resultado de la consulta anterior sin el registro eliminado, en el área de **Consulta Presupuestos**.

Gestión de clientes

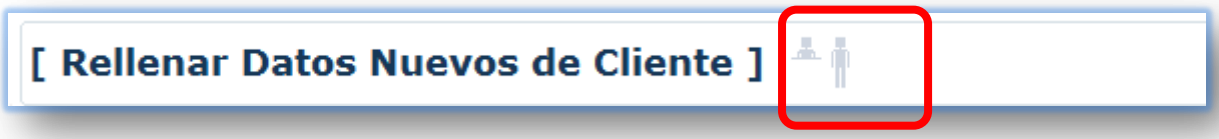


Pantalla para gestionar Clientes.

El sistema permitirá la gestión (alta y consulta) de dicha información.

Alta

Al pulsar sobre el enlace



se mostrará una pantalla desde la que se podrá dar de alta a un nuevo cliente.

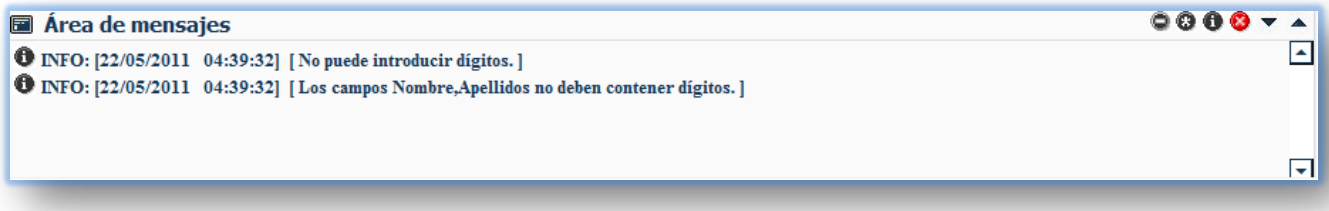
En la parte central se abrirá un formulario solicitando la siguiente información (los campos obligatorios para el usuario se identificarán con el color amarillo):

Y contendrá los siguientes botones:

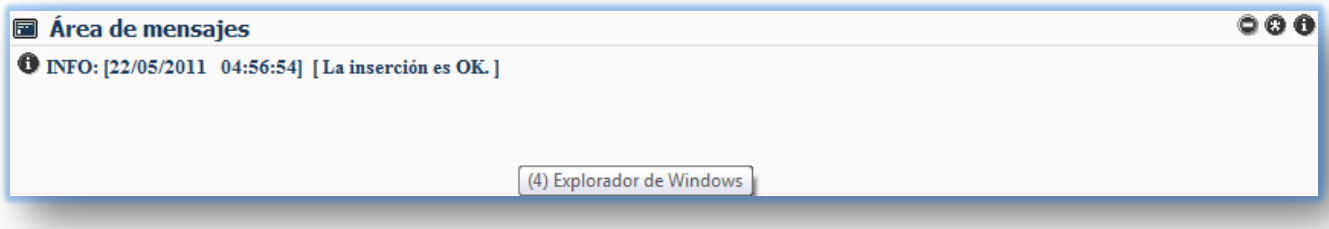
Disquete  al hacer clic validará los campos del formulario, teniendo en cuenta los campos obligatorios y validaciones del NIF, fechas…, una vez realizadas éstas se procederá a insertar en Base de datos el contenido del formulario.

Si se pulsa sobre este icono sin haber introducido todos los datos se mostrará un aviso en el área de mensajes de la página indicándolo, puesto que éstos son campos obligatorios en la inserción.

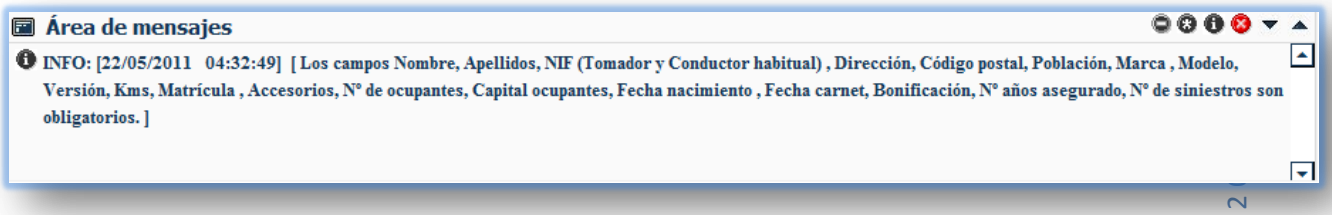
Si los campos obligatorios introducidos no son correctos la aplicación nos advierte en el área de mensajes, que los campos no son los correctos, es decir se valida que cada campo sea el correcto.



Si la inserción en Base de Datos ha sido satisfactoria, se mostrará por pantalla dicha información en la **Consulta Clientes**, permaneciendo además los datos en la parte del frame superior.



Este procedimiento inserta en la Tabla uoc.cliente , pero esto sólo si el cliente no existiera .

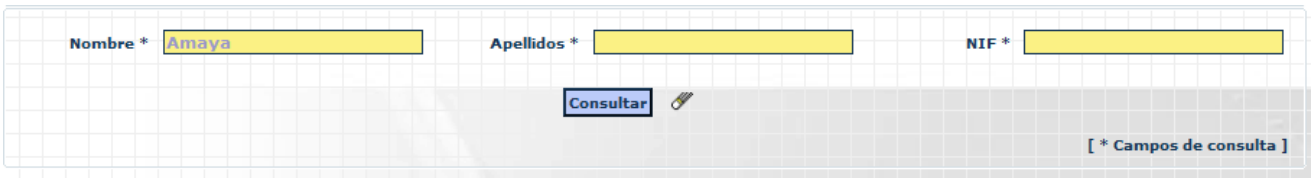


Goma de borrar  al hacer clic borra los campos del formulario.

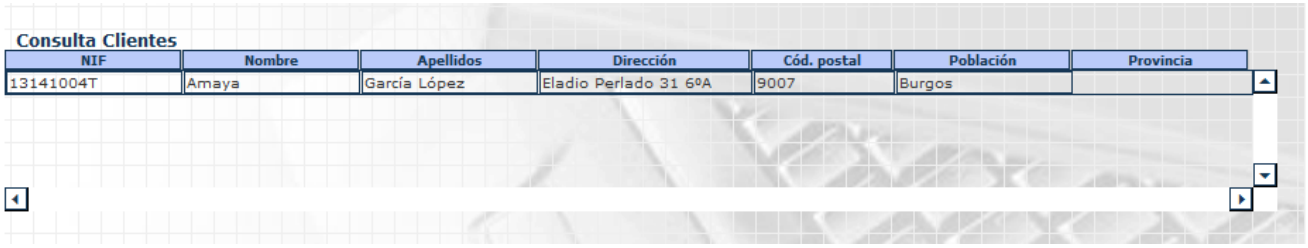
Este icono  al hacer clic cierra el formulario Datos Presupuesto Auto.

Consulta

Desde la pantalla de alta se podrá realizar una consulta introduciendo cualquiera de los campos que se muestran y pulsando sobre el botón Consultar.

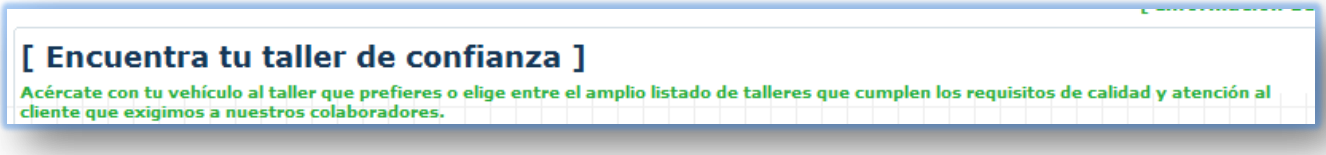


Como resultado, y tras a acceder a la tablas **uoc.cliente** con los datos introducidos para obtener la información, se mostrará en la parte inferior denominada **Consulta Clientes** un listado con la consulta obtenida.



Información

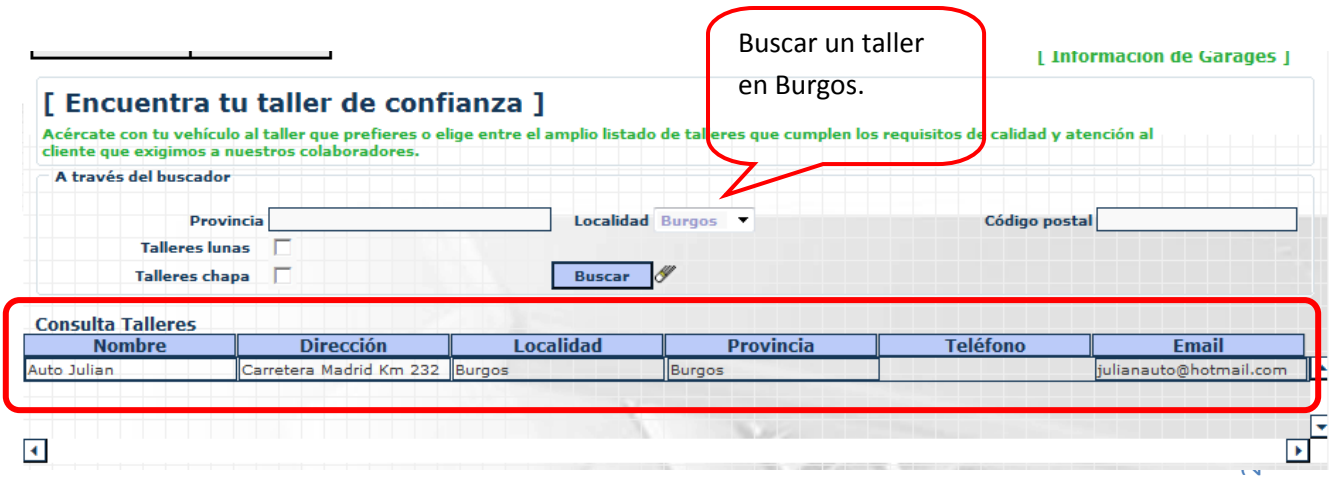
Esta pantalla es meramente informativa, el colaborador o el jefe de producción puede informar al cliente de donde encontrar el garaje más cercano a sus necesidades.



Consulta

Desde la pantalla principal se podrá realizar una consulta introduciendo cualquiera de los campos que se muestran y pulsando sobre el botón Buscar.

Como resultado, y tras a acceder a la tablas **uoc.informacion** con los datos introducidos para obtener la información, se mostrará en la parte inferior denominada **Consulta Talleres** un listado con la consulta obtenida.

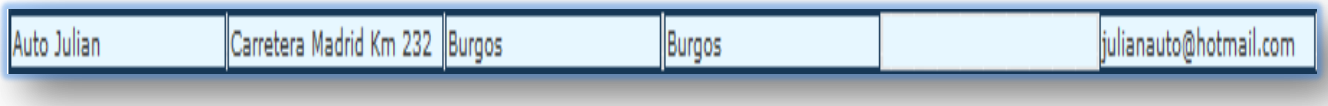


Para que podamos ver un taller se ha introducido esta sentencia en Base de datos:

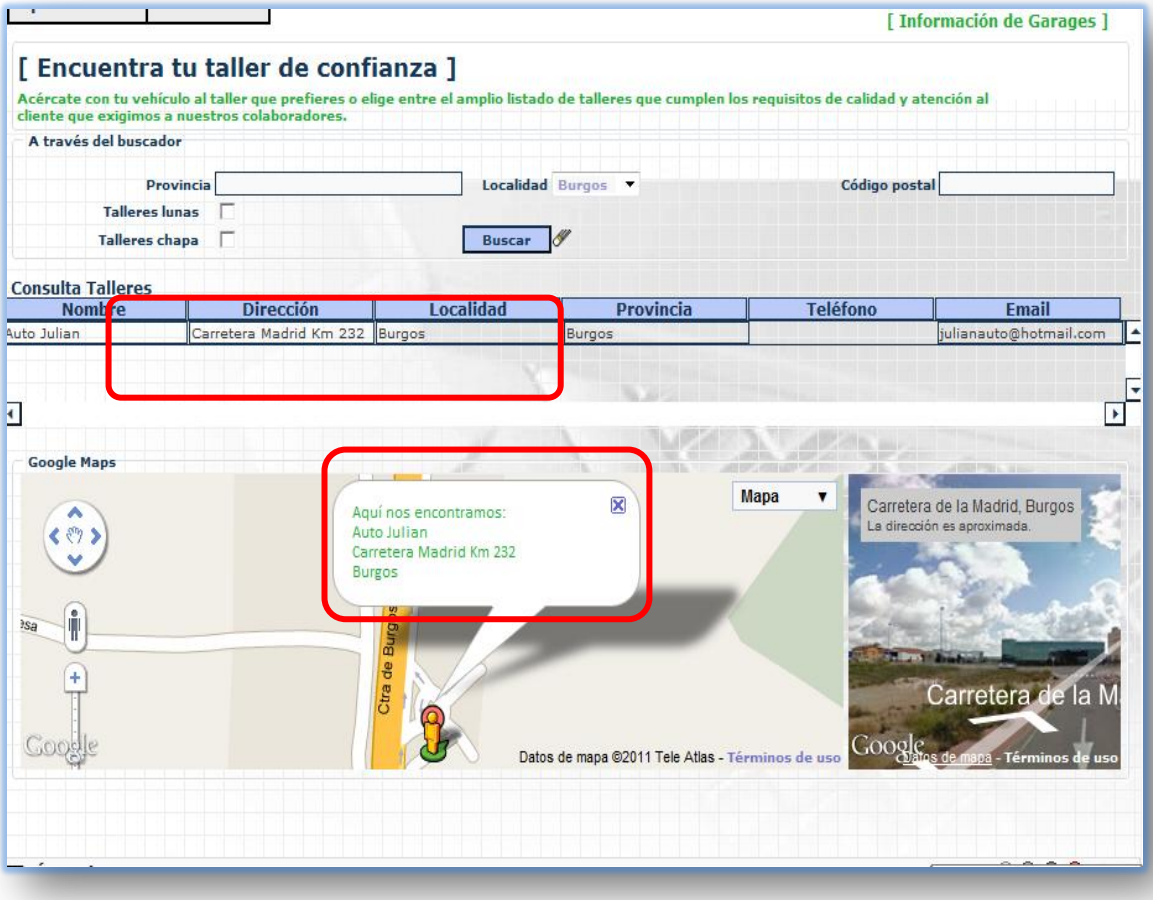
INSERT INTO `uoc`.`informacion` (`idInformacion`, `nombre`, `direccion`, `codigopostal`, `ciudad`, `provincia`, `pais`, `email`, `tipo`, `movil`) VALUES ('2G', 'Auto Julian ', 'Carretera Madrid Km 232', '09006', 'Burgos', 'Bugos', 'España', 'julianauto@hotmail.com', 'lunas', '666123456');

Como novedad se ha querido implementar el poder visualizar en **GOOGLE MAPS** la dirección del Taller buscado.

Para poder visionar la dirección una vez filtrado el taller que buscamos y pulsado el botón buscar, debemos seleccionar la fila correspondiente al taller que queremos y dar un click sobre ésta y se producirá la búsqueda requerida.



De esta manera visionaremos en Google Maps la dirección exacta del taller



Para utilizar la Aplicación web solamente entrar en el Navegador Internet Explorer 8 ó 9

http://localhost:8080/tfc-uoc