

Índice

1. Introducción.....	1
2. Objetivos Generales.....	3
2.1 Objetivos funcionales	3
2.2 Objetivos futuros	4
2.3 Razones para escoger este proyecto	4
3. Planificación.....	7
3.1 Análisis de costes	8
4. Análisis.....	11
4.1 Análisis de requerimientos	11
4.2 Modelo de casos de uso	12
4.3 Actores principales	13
4.4 Casos de uso	13
5. Diseño de la aplicación.....	23
5.1 Diseño de la capa dominio	25
5.1.1 Diseño de la clase Escuela.....	26
5.1.2 Diseño de la clase Sector.....	27
5.1.3 Diseño de la clase Usuario	28
5.1.4 Diseño de la clase ruta.....	29
5.1.5 Diseño de la clase RutaHistorial.....	30
5.2 Diseño de la capa persistencia.....	31
5.3 Diseño de la capa aplicación	32
5.4 Diseño de la base de datos.....	33
6. Tecnologías utilizadas e implementación.....	35
6.1 JavaServerFaces	35
6.1.1 Características principales	35
6.1.2 Modelo.....	37
6.1.3 Vista.....	38
6.1.4 Controlador.....	38
6.1.5 Convertir y validar datos	40
6.1.6 Apache MyFaces	43
6.2 iBatis.....	45
6.2.1 Implemetación iBatis.....	46
7. Conclusiones.....	51
8. Bibliografía.....	53
9. Glosario	55
10. Anexo	57
10.1 Tutorial netBeans VisualWeb JavaServerFaces.....	57
10.2 Como conectar la PDA al servidor apache.....	63
10.3 Contenido de los CD's.....	63

2 Introducción

1. Introducción

Internet avanza a pasos agigantados por la historia, se ha convertido en la fuente de información más grande creada por el hombre, desde sus inicios alrededor de los años sesenta, hasta hoy, ha pasado poco más de medio siglo, que en comparación con la vida humana en la tierra, la proporción es ínfima. Pero aun es más pequeña si la comparamos desde que Internet llegó a los hogares, es más, aunque nos cueste creer hay muchos hogares donde aun no ha llegado Internet. No creo que nadie sepa cual será el camino que seguirá la red de redes, lo que si sabemos es que los cambios en ella son continuos y es tan enorme, que seguramente ha superado con creces las aspiraciones de sus creadores. En este proyecto la intención es aprovechar la red de redes, para organizar la información relacionada con las rutas de escalada y facilitar la gestión personal, y aprovechar las tecnologías que nos rodean en nuestro beneficio.

Como punto de partida diré que hace años que escalo, no soy un gran escalador, pero disfruto escalando y superando mis miedos día a día. Nunca he tenido el privilegio de salir de España y escalar por el mundo, algún día cuando disponga de más tiempo lo intentaré, si más no hay que destacar que España y en especial Cataluña posee paredes que se han convertido en paraísos de reconocimiento mundial y visitados por los mejores escaladores del mundo. Como es el caso de Siurana, Montserrat, Vilanova De Meia... y muchas otros lugares maravillosos, sin ir más lejos, seguramente el que es considerado mejor escalador del mundo (Chris Sharma, americano), ha fijado su residencia en Lérida.

Volviendo a la idea del proyecto, no tenia demasiado claro que tipo de aplicación podía realizar, y dándole vueltas a la cabeza se me ocurrió ManagementClimbingRoute, que simplemente es una forma de ayudarme a gestionar las rutas que realizo y quien sabe si quizás algún también ayude a gestionar la rutas realizadas a otra persona.

2. Objetivos Generales

Los objetivos de este proyecto son los siguientes:

- Conseguir crear una aplicación que gestione la información de las bases de datos mediante un framework, concretamente el marco de trabajo iBatis.
- Utilizar un sistema gestor de bases de datos no utilizado por mi hasta la realización del proyecto, concretamente MySQL
- Aprender a utilizar una nueva tecnología para la realización del proyecto, la tecnología JavaServerFaces.
- Conseguir que la aplicación sea funcional desde el navegador de una PDA.
- Crear dos tipos de usuarios, el usuario registrado y el invitado que tendrá unas funcionalidades más limitadas.

2.1 Objetivos funcionales

Para obtener una aplicación funcional el usuario y el invitado deben poder realizar las siguientes acciones,

- Deben poder observar las rutas existentes en la base de datos
- Deben poder darse de alta en la aplicación y en caso de considerarlo necesario darse de baja
- Es necesario que pueda modificar sus datos existentes en la base de datos.
- El usuario registrado debe poder añadir rutas a su historial de rutas
- El usuario registrado debe poder ver las características de las rutas de su historial
- Deben poder buscar rutas en función de la escuela y el sector.
- El usuario registrado debe poder eliminar una ruta de su historial si lo considera oportuno.
- Solicitar su contraseña en caso de pérdida.
- El usuario registrado debe disponer de su ámbito de sesión.

2.2 Objetivos futuros

No se había planteado nada en referencia a objetivos futuros de la aplicación, pero ha medida que la aplicación ha ido avanzando se ha visto la posibilidad de realizar mejores sustanciales en un futuro cercano. Veamos algunas mejoras que se podrían aplicar:

- Actualmente no existe un usuario administrador que controle la información de la aplicación y decida introducir nuevas rutas, escuelas o sectores, sería interesante crearlo para facilitar la ampliación de la información. El usuario administrador accede directamente al ordenador donde se encuentra el sistema gestor de bases de datos.
- Al crear el usuario administrador, es necesario ampliar los casos de uso, los nuevos casos de uso más destacables serían; crear escuela, crear sector, crear ruta, sus respectivas modificaciones, y eliminaciones.
- También sería muy interesante la creación de una tienda de material de escalada en la misma aplicación. Esta opción complicaría mucho la aplicación, pero sería un opción muy recordable.
- Aplicar a todo el conjunto de la aplicación estilos CSS con la ayuda de un diseñador web que permitieran hacerla visualmente más agradable.
- Encriptar la información de la base de datos para prevenir ataques, aunque la información no se considera relevante para ser propensa a ellos.

2.3 Razones para escoger este proyecto

La primera vez que leí el título de este proyecto, tenía una idea equivocada del significado del mismo. Pensaba, que con este proyecto entraría en un mundo desconocido y extraño para mí, la programación de aplicaciones para dispositivos móviles con J2ME. Esta idea me gustaba y me gusta, siempre he creído en el potencial de estos dispositivos. Pero más adelante la idea que tenía sobre el proyecto cambio radicalmente, se transformó en una aplicación para PDA, que se ejecuta a través del navegador en este caso Internet Explorer. Esta transformación de la idea no me desagradó, es más me ha permitido conocer frameworks que hasta hace unos meses eran completos desconocidos para mí. El

framework iBatis para trabajar con la persistencia y JavaSeverFaces para implementar y estructurar la aplicación con el modelo vista controlador. Si leemos desde la distancia estas palabras comprenderemos que la principal razón para escoger este proyecto es la necesidad de aprender algo nuevo y demostrarme que era capaz de hacerlo.

3. Planificación

La planificación y la metodología seguidas para realizar el proyecto ha sido la siguiente, primero surgió la idea de la aplicación ManagementClimbingRoute, es decir el gestor de rutas de escalada realizadas, una vez la idea empezó a madurar, se diseñó el modelo del dominio, los objetos que serían necesarios, en nuestro caso ruta, usuario, sector y escuela. Una vez decididos los objetos y diseñado el dominio, se creó la base de datos con el sistema gestor de base de datos MySQL, poco después de diseñar la base de datos y tenerla creada, se empezaron a diseñar los casos de uso, los casos de uso a realizar y los casos de uso a realizar en un futuro de la aplicación, este se comenta también en los objetivos futuros. Una vez decididos los casos de uso, se empezó a forjar el diseño de la arquitectura de la aplicación, que patrones se iban a utilizar, en este caso el modelo vista controlador y la factoría con singleton. También se diseñaron los paquetes necesarios. Una vez diseñada la arquitectura de la aplicación se decidió que tecnologías se iban a utilizar, no había duda que el lenguaje era java, eso se sabía desde el principio, pero la decisión de utilizar JavaServerFaces y iBatis no estaba demasiado claro, básicamente porque al principio se consideró la posibilidad de utilizar J2ME y otros framework de base de datos como openBaseMovil o Floggy que fueron descartados, porque se consideraron demasiado complicados y de funcionalidad dudosa, en otras palabras más sinceras no se consiguió entender del todo su funcionamiento.

Con esto claro se empezaron a implementar los casos de uso y apareció el primer problema, faltaba un objeto del dominio, el historialRutas, el cual permite separar las rutas existentes de las rutas introducidas por el creador de la aplicación, se podía encapsular todas en el mismo objeto ruta, pero al almacenar en la base de datos era bastante caótico. Con este cambio en el dominio y la base de datos, permitimos que la entidad ruta de la base de datos ruta, solo contenga las rutas existentes, sin que estén duplicadas en la entidad, mientras que en historialRutas se almacenan las rutas realizadas por cada usuario, duplicadas pero cada una asociada a un usuario. Una vez todos los casos de uso implementados, se fue recopilando la información para redactar la memoria del proyecto. Finalmente empezó el testing de la aplicación via web y PDA.

3.1 Análisis de costes

En este apartado se van a calcular los costes del proyecto. Los costes de la aplicación se dividen en costes hardware, software e implementación del proyecto. Empezaremos por los costes de hardware, el ordenador tiene las siguientes características:

Procesador Intel Dual Core E5200 2,5Ghz
 Placa Base Asus P5GC-MX/1333 Socket 775
 Disco Duro 500GB SATA Western Digital/Seagate
 Memoria Kingston 4GB DDR2 800
 Tarjeta gráfica 8400GS 512MB PCI-e
 Monitor LCD 20"

El software necesario para llevar acabo la aplicación, dado que MySQL y el IDE netbeans son gratuitos no es necesario imputarlo al proyecto, en todo caso de debería imputar el precio del WindowsXP, ya que es necesario para el funcionamiento del ordenador y además la aplicación ha sido provada en el navegador Internet Explorer. El precio del ordenador asciende a 468 euros, en los que se incluye también el sistema operativo WindowsXP con el navegador Internet Explorer.

El precio de una PDA como por ejemplo el modelo Dell Axim X50v, con Windows Mobile incorporado es de 450euros. También hay que imputar la tarifa de conexión a Internet en esta caso tarifa plana de 40euros al mes.

A todo esto solo nos falta añadir el coste de la implementación, si el proyecto tiene una durada de 16 semanas, y el programador cobra 18 euros/hora, contando una dedicación de 3 horas/día, y 5 días a la semana es coste de implementación será, 240horas por 18euros/hora, que suman un total de 4320 euros. En la siguiente tabla se contempla el cote total de la aplicación.

Ordenador	PDA	Internet	Implementación	Total
468€	450€	160€	4320€	5398€

Se ha de tener en cuenta todo el tiempo invertido en formación, búsqueda y aprendizaje, es decir la curva de aprendizaje, otra persona con experiencia en este tipo de proyectos,

conseguiría en el mismo proyecto en menos tiempo, y eso se vería reflejado en los costes.

4. Análisis

Conseguir un buen análisis del producto software es la primera etapa para su correcta creación, en este caso el cliente y el desarrollador son la misma persona, la captura el análisis y la especificación de requisitos es crucial, para lograr los objetivos propuestos. Gran parte del éxito de un proyecto de software radicará en la identificación de las necesidades del negocio (definidas por la alta dirección), así como la interacción con los usuarios funcionales.

4.1 Análisis de requerimientos

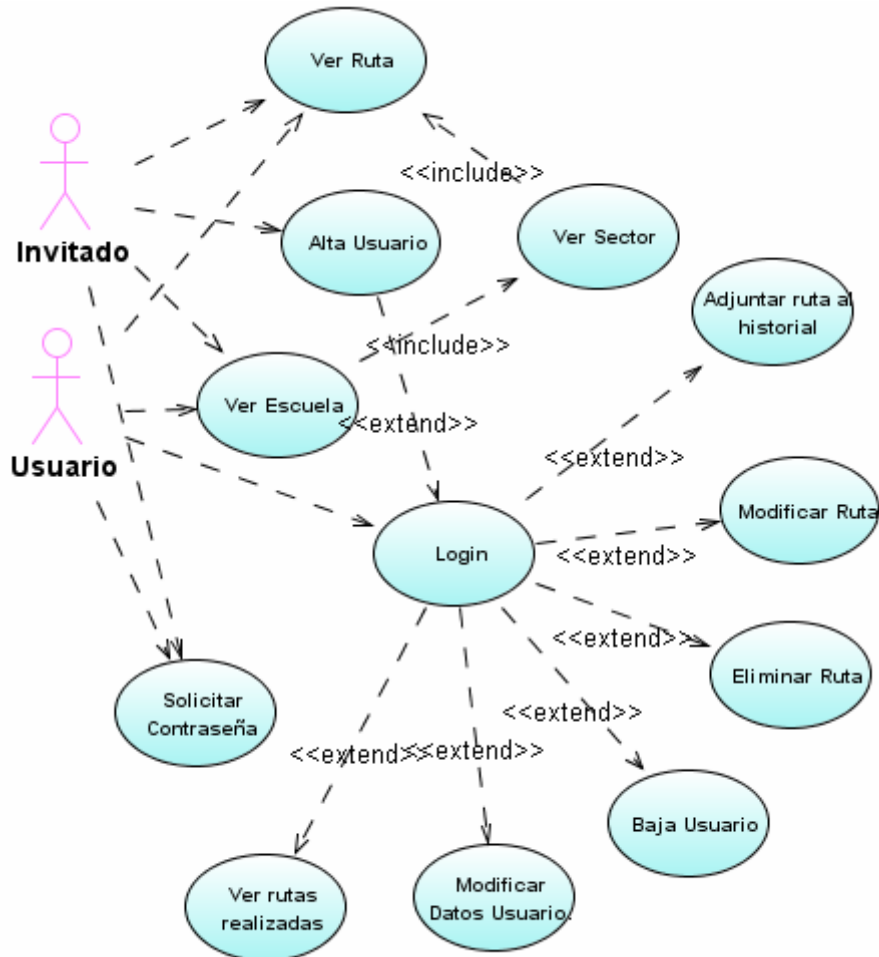
Para realizar cualquier proyecto que tengamos entre manos es prioritario hacer un buen análisis de requerimientos, para no tener que derruir el trabajo realizado en un futuro, o como se suele decir desandar lo andado. En este proyecto el objetivo principal es que el usuario de la aplicación pueda gestionar las rutas que realiza . Poco a poco se va despedazando la idea general para ir complementándola con pequeñas ideas que van haciendo el proyecto más y más grande, y se podría decir que así se llegan a crear proyectos de envergaduras inmensas.

El proyecto ManagementClimbingRoute ha sido modificado demasiadas veces, y seguramente en un futuro volverá a ser modificado, antes he mencionado el objetivo del proyecto, pero ahora cerca del final, desearía incorporarle más funcionalidades, que serán comentadas más adelante.

Una vez se tienen especificados y definidos los objetivos del proyecto, se empieza a plantear a quien irá dirigida la aplicación, en este caso a cualquier persona que le guste realizar rutas de escalada, a priori en tierras ibéricas, es posible que más adelante y ampliando la aplicación se puedan introducir rutas de cualquier país. Se perfilan dos tipos de usuario, el usuario invitado y el usuario registrado, el primero básicamente solo puede navegar por la aplicación sin modificar nada, ni crear rutas en su historial de rutas, ni eliminar, hasta no estar registrado, el usuario registrado podrá acceder a una variedad más elevada de funcionalidades. En los siguientes apartados se describen los casos de uso, que ayudaran a comprender mejora la aplicación.

4.2 Modelo de casos de uso

En el siguiente diagrama UML de casos de uso podemos ver todo lo que puede realizar el Invitado y el Usuario.



En el diagrama de casos de uso superior, se observan los diferentes casos de uso a implementar, se consideró la posibilidad de crear un caso de uso gestión rutas, que incluyera gran parte de los mencionados, como por ejemplo ver ruta, adjuntar ruta historial, eliminar ruta... Pero se decidió especificarlos por separado, ya que no existía el usuario administrador en la aplicación, en un futuro de la aplicación si el usuario administrador accede a la aplicación para crear nuevas rutas, escuelas y sectores se tendría que modificar el diagrama. Pero en la actualidad las rutas, los sectores y las escuelas son introducidos por el creador directamente en la base de datos.

Para realizar la aplicación es necesario analizar cada uno de los casos de uso, y completar las funcionalidades descritas, en esta aplicación se distinguen dos actores, el invitado que puede, ver ruta, ver escuela que integra ver sector y ver rutas de ese sector,

también puede solicitar contraseña, aunque no la obtendrá porque no está dado de alta como usuario, y finalmente se puede dar de alta, y automáticamente quedar logado, a partir de esa acción acceder a las mismas funcionalidades que un usuario registrado. El usuario, puede acceder a todas las funcionalidades que accede el invitado, exceptuando el alta usuario, al estar ya dado de alta. A parte de esas funcionalidades también puede acceder a las siguientes, adjuntar ruta al historial, modificar ruta, eliminar ruta del historial, darse de baja como usuario y ver rutas realizadas, más adelante se explicará cada caso de uso con detalle, para una mejor comprensión.

4.3 Actores principales

En esta aplicación solo entran en juego dos actores, el invitado y el usuario, la única diferencia que existe entre ellos es el grado de implicación respecto a la aplicación, si un invitado considera que le puede ser útil, se dará de alta como usuario i disfrutará de todas la funcionalidades, en caso contrario solo podrá utilizar las funcionalidades mencionadas anteriormente.

4.4 Casos de uso

Ver ruta

- **Objetivos:** Se accede para poder ver las características de una ruta.
- **Descripción:** El usuario o el invitado acceden y el sistema facilita el acceso.
- **Actor principal:** el usuario, el invitado.
- **Precondiciones:** El sistema esta en funcionamiento..
- **Flujo normal**
 1. El usuario accede a la aplicación.
 2. El sistema le muestra las rutas existentes.
 3. El usuario o invitado escoge la ruta que desea ver con más detalle.
 4. El sistema le muestra la ruta.
- **Flujo alternativo**
 - 3.1. El usuario o invitado desean escoger otra ruta

3.1.1. El sistema le da la opción de volver a escoger otra ruta volviendo al punto 2.

- **Poscondiciones:** Ninguna.

Ver escuela

- **Objetivos:** Se accede para poder ver las características de una escuela.
- **Descripción:** El usuario o el invitado acceden y el sistema facilita el acceso.
- **Actor principal:** el usuario, el invitado.
- **Precondiciones:** El sistema esta funcionamiento.
- **Flujo normal**
 1. El usuario o invitado acceden a la aplicación.
 2. El sistema le muestra las escuelas existentes.
 3. El usuario o invitado escoge la escuela que desea ver con más detalle.
 4. El sistema le muestra la escuela escogida.
- **Flujo alternativo**
 - 1.1. El usuario o invitado desean ver otra escuela.
 - 1.1.1. El sistema le permite escoger una nueva escuela volviendo al punto 2.
- **Poscondiciones:** Ninguna.

Ver sector

- **Objetivos:** Se accede para poder ver las diferentes rutas del sector
- **Descripción:** El usuario o el invitado acceden y el sistema facilita el acceso.
- **Actor principal:** el usuario, el invitado.
- **Precondiciones:** el usuario o el invitado deben haber realizado el caso de uso ver escuela.
- **Flujo normal**
 1. El sistema muestra los diferentes sectores a escoger.
 2. El usuario o el invitado escogen un sector
 3. El sistema les muestra las rutas existentes en ese sector.

- **Flujo alternativo**

- 2.1. El usuario decide escoger otro sector

- 2.1.1. El sistema le da la opción de retroceder y escoger otro sector.

- **Poscondiciones:** Ninguna.

Caso de uso alta usuario

- **Objetivos:** Se accede para poder disfrutar de la totalidad de casos de uso.
- **Descripción:** El invitado pasa a ser usuario registrado y el sistema le permite acceder al resto de casos de uso.
- **Actor principal:** El invitado.
- **Precondiciones:** El sistema esta en funcionamiento.
- **Flujo normal**
 1. El invitado accede a la aplicación.
 2. La aplicación muestra en todo momento la opción de darse de alta.
 3. El sistema le muestra el formulario a rellenar.
 4. El invitado rellena el formulario y envía la información.
 5. El sistema hace las comprobaciones de los datos introducidos.
 6. El sistema comunica al usuario que el proceso se ha realizado con éxito.
- **Flujo alternativo**
 - 5.1. Los datos introducidos no son correctos.
 - 5.1.1. El sistema le comunica al invitado que hay errores y vuelve al punto 3.
- **Poscondiciones:** El invitado ha pasado a ser usuario registrado, ahora está registrado en la base de datos. Ya puede acceder a los casos de uso que no podía acceder como invitado, crear ruta, modificar ruta, eliminar ruta, baja usuario, ver ruta realizada.

Solicitar contraseña

- **Objetivos:** Enviar al usuario su contraseña.
- **Descripción:** El usuario solicita la contraseña que será enviada a su correo electrónico.
- **Actor principal:** El usuario.
- **Precondiciones:** El sistema está en funcionamiento.
- **Flujo normal**
 1. El usuario accede a la aplicación.
 2. La aplicación mostrará en todo momento la opción de solicitar contraseña.
 3. El sistema muestra un formulario a rellenar.
 4. El usuario introduce los datos.
 5. El sistema valida los datos.
 6. El sistema envía la contraseña.
 7. El sistema comunica que el proceso se ha realizado con éxito.
- **Flujo alternativo**
 - 5.1. El usuario no existe en la base de datos.
 - 5.1.1. El sistema le comunica que no existe en la base de datos, se muestra la opción de darse de alta.
- **Poscondiciones:** Ninguna.

Login

- **Objetivos:** El usuario demuestra su existencia en la base de datos.
- **Descripción:** El usuario se autentifica para poder acceder a los casos de uso donde estar registrado es obligatorio.
- **Actor principal:** El usuario.
- **Precondiciones:** Haberse dado de alta con anterioridad y que el sistema este en funcionamiento.
- **Flujo normal**
 1. El usuario accede a la aplicación
 2. El sistema le muestra la opción de logarse.

3. El usuario rellena el formulario y envía los datos.
4. El sistema comprueba los datos introducidos.
5. El sistema muestra las opciones de un usuario logado.

- **Flujo alternativo**

- 4.1. Los datos introducidos no son correctos

- 4.1.1. El sistema comunica al usuario que no existe en el registro de la base de datos

- 4.1.2. El sistema vuelve al punto 2.

- **Poscondiciones:** Ninguna.

Adjuntar ruta en el historial

- **Objetivos:** El usuario desea introducir una nueva ruta a su historial.
- **Descripción:** El usuario accede para registrar una nueva ruta.
- **Actor principal:** El usuario.
- **Precondiciones:** El usuario debe haberse autenticado o haber realizado el proceso de alta de usuario.

- **Flujo normal**

1. El sistema le muestra las opciones a escoger de la aplicación.
2. El usuario selecciona la opción adjuntar ruta en el historial.
3. El sistema le muestra los datos a rellenar.
4. El usuario introduce los datos y selecciona validar.
5. El sistema comprueba que los datos estén correctamente.
6. El sistema le comunica que el proceso ha sido correcto.

- **Flujo alternativo**

- 5.1. Los datos introducidos por el usuario no son correctos.

- 5.1.1. El sistema comunica al usuario que los datos no son correctos y vuelve al punto 3.

- **Poscondiciones:** El usuario ha adjuntado una nueva ruta en su historial de rutas.

Modificar ruta

- **Objetivos:** El usuario desea cambiar algún atributo de la ruta.
- **Descripción:** El usuario accede a la aplicación para modificar una ruta creada con anterioridad.
- **Actor principal:** El usuario.
- **Precondiciones:** El usuario se ha autenticado y ha adjuntado alguna ruta a su historial.
- **Flujo normal**
 1. El sistema le muestra las opciones a escoger de la aplicación.
 2. El usuario escoge selecciona de modificar una ruta.
 3. El sistema muestra las rutas adjuntadas por el usuario.
 4. El usuario escoge una ruta a modificar.
 5. El sistema le muestra la información de la ruta.
 6. El usuario modifica los datos de la ruta y los envía.
 7. El sistema comprueba que los datos sean correctos.
 8. El sistema comunica al usuario que la modificación se ha producido con éxito.
- **Flujo alternativo**
 - 7.1 Los datos introducidos no son correctos.
 - 7.1.1. El sistema le comunica al usuario que los datos no son correctos, y vuelve al punto 5.
- **Poscondiciones:** El usuario ha modificado una ruta de su historial.

Eliminar ruta

- **Objetivos:** Se accede para poder eliminar una ruta del historial.
- **Descripción:** El usuario decide eliminar una ruta de su historial.
- **Actor principal:** El usuario.
- **Precondiciones:** El usuario se debe de haber autenticado y adjuntado como mínimo una ruta en su historial.

- **Flujo normal**
 1. El sistema le muestra las opciones a escoger.
 2. El usuario escoge la opción de eliminar una ruta.
 3. El sistema le muestra las rutas.
 4. El usuario escoge las rutas a eliminar.
 5. El usuario confirma la eliminación.
 6. El sistema elimina las rutas.
 7. El sistema vuelve al punto 3.
- **Flujo alternativo**
 - 5.1. El usuario no confirma la eliminación
 - 5.1.1. El sistema vuelve al punto 3.
- **Poscondiciones:** El usuario ha eliminado las rutas del historial.

Baja usuario

- **Objetivos:** Se accede para darse de baja de la aplicación.
- **Descripción:** El usuario accede a la aplicación y el sistema le facilita la opción.
- **Actor principal:** El usuario.
- **Precondiciones:** El usuario de haberse dada de alta o autenticado con anterioridad.
- **Flujo normal**
 1. El sistema le muestra las opciones a escoger.
 2. El usuario escoge darse de baja.
 3. El sistema muestra la información con los datos del usuario y pide la confirmación.
 4. El usuario confirma que desea darse de baja.
 5. El sistema elimina al usuario.
- **Flujo alternativo**
 - 4.1. El usuario no confirma la baja de la aplicación.

4.1.2. El sistema le muestra las opciones de la aplicación.

- **Poscondiciones:** El usuario ha sido eliminado de la base de datos mediante un borrado lógico.

Ver rutas realizadas

- **Objetivos:** El usuario accede par ver las rutas realizadas.
- **Descripción:** El usuario accede a la aplicación y el sistema le facilita el acceso.
- **Actor principal:** El usuario.
- **Precondiciones:** El usuario se autentifica o se da de alta en la aplicación.
- **Flujo normal**
 1. El sistema le ofrece las opciones a escoger.
 2. El usuario escoge la opción de ver las rutas realizadas.
 3. El sistema le muestra las rutas realizadas.
- **Flujo alternativo**
 - 2.1. El usuario no escoge la opción de ver rutas realizadas.
 - 2.1.1. El sistema le muestra las opciones a escoger.
- **Poscondiciones:** Ninguna.

Modificar usuario

- **Objetivos:** El usuario accede para modificar sus datos del registro de la base de datos.
- **Descripción:** El usuario accede a la aplicación y el sistema le facilita el acceso.
- **Actor principal:** El usuario.
- **Precondiciones:** El usuario se autentifica o se da de alta en la aplicación.
- **Flujo normal**
 1. El sistema le ofrece las opciones a escoger.
 2. El usuario escoge la opción de modificar sus datos personales.
 3. El sistema muestra los datos del usuario.
 4. El usuario modifica los datos y los envía.
 5. El sistema comprueba que la información introducida sea correcta.

6. El sistema informa al usuario que el proceso se ha llevado a cabo exitosamente.

- **Flujo alternativo**

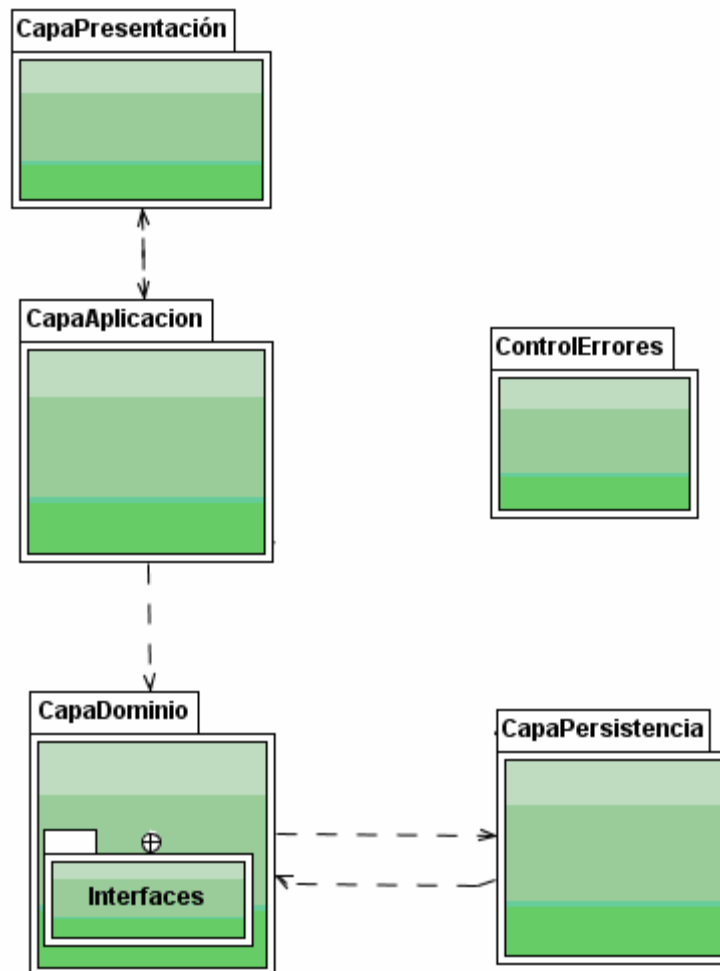
- 5.1. El sistema informa al usuario que la información introducida no es correcta.

- 5.1.1. El sistema vuelve al punto 3.

- **Poscondiciones:** El usuario ha modificado sus datos personales en el registro de la base de datos.

5. Diseño de la aplicación

En este capítulo del proyecto se explicará el diseño de la aplicación a partir de herramientas visuales como los diagramas de clases. El framework o marco de trabajo VisualWebJavaServerFaces utiliza el patrón Modelo Vista Controlador, si observamos el diagrama de paquetes, comprobaremos las siguientes equivalencias, la capa Presentación hace referencia a la Vista, la capa Aplicación al Controlador, aunque en JavaServerFaces se utilice un controlador monolítico, en nuestro caso comprobaremos más tarde que poseemos un controlador para cada clase de la capa Dominio, y Modelo equivale a la capa Dominio de la aplicación.



Tal y como está estructurada la aplicación posee el patrón Modelo Vista Controlador y el patrón Capas, de esta manera obtendremos una aplicación más sostenible. Por ejemplo podemos modificar requerimientos de la aplicación que afecten a la capa Persistencia sin preocuparnos del efecto sobre otras capas de la aplicación, incluso podríamos cambiar el sistema gestor de bases de datos sin que afectase a la capa Dominio.

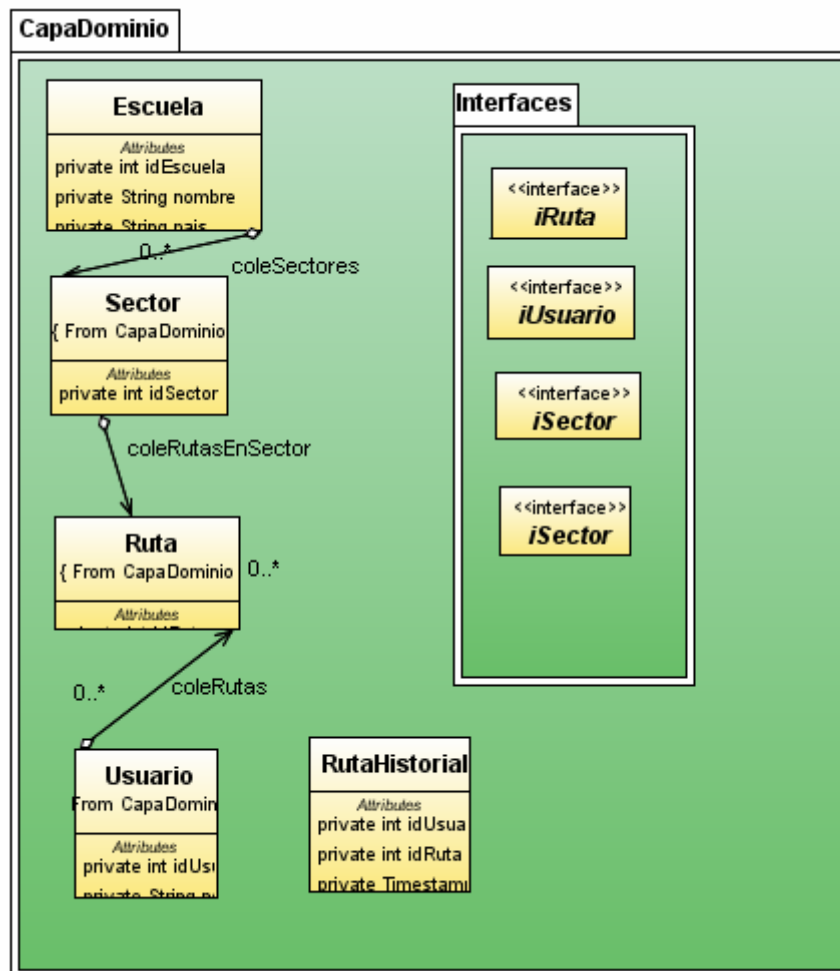
El acoplamiento de la aplicación es bastante claro, la capa Presentación no podrá acoplarse directamente con la capa Dominio sin pasar antes por la capa Aplicación, la capa Persistencia deberá pasar por la capa Dominio (en nuestro caso las interfaces del dominio) para que la capa Aplicación pueda obtener información de la base de datos. De esto podemos deducir que la capa Dominio se mantiene aislado del resto, bueno siendo atacado por la capa Aplicación y la capa Persistencia, conservando un equilibrio sostenible entre la lógica del negocio y la aplicación.

Debido a la localización de las interfaces en el Dominio, siempre que la Aplicación desee hablar con la Persistencia, deberá pasar por el Dominio. Por último, la Persistencia se acoplará con el Dominio para montar los objetos. **Para crear una aplicación más sostenible se decidió crear unas interfaces en el Dominio, que son implementadas en la persistencia generando un acoplamiento débil, por esta razón no existe un acoplamiento fuerte entre la capa aplicación y la capa persistencia.**

5.1 Diseño de la capa dominio

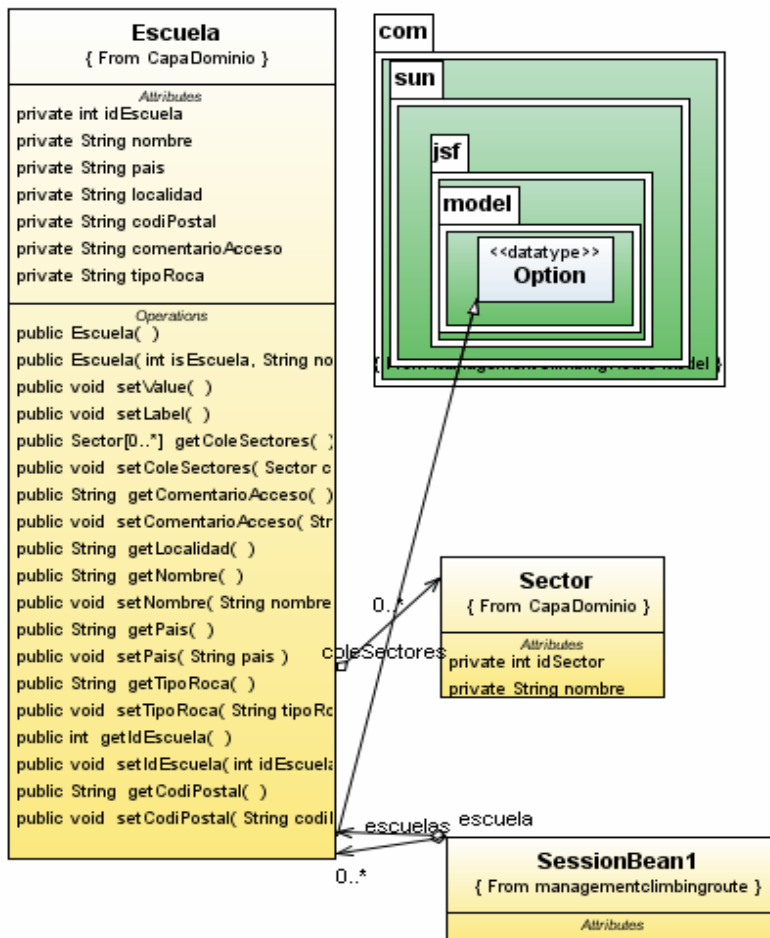
Esta capa contiene los objetos que forman el Dominio, tiene como función principal la encapsulación de la información o datos necesarios para cada objeto, que a medida que se necesiten se distribuirán a la presentación o la persistencia dentro del Dominio también encontramos el paquete de interfaces, que se encargan de acoplar el Dominio y la capa aplicación con la capa persistencia.

La clase Escuela posee una colección de objetos del tipo Sector, la clase Sector posee a su vez una colección objetos del tipo ruta, y el usuario posee una colección de objetos del tipo de ruta, veamos el diagrama.



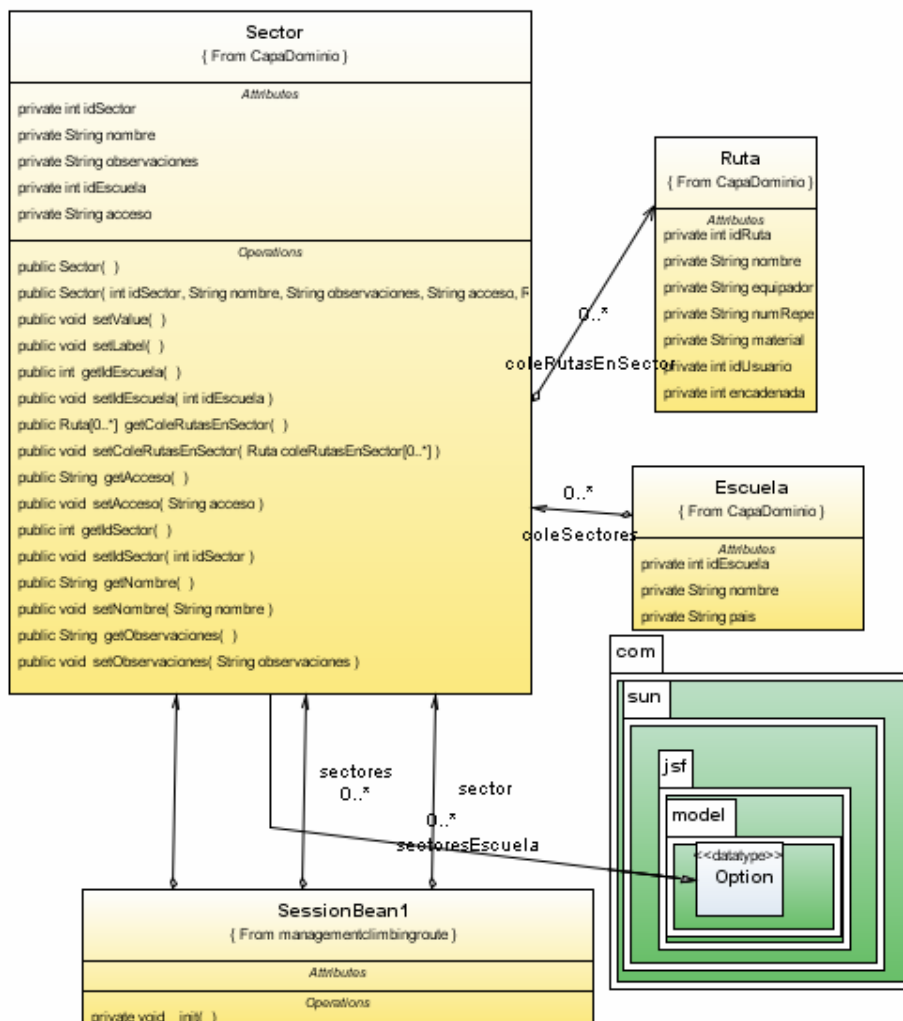
5.1.1 Diseño de la clase Escuela

En el siguiente diagrama podemos observar con que otras clases interactúa la clase Escuela, existe un atributo en la clase SessionBean1, del tipo escuela que nos permite mostrar información referente a la misma, o extraer información de objetos como sector o ruta que están relacionados con escuela, como hemos mencionado antes la clase Escuela posee una colección (ArrayList) de sectores, y a su vez el sector posee otra colección de rutas. También hay que destacar que la SessionBean1 no ataca directamente a la clase escuela si desea obtener información o datos de la misma pasa antes por el ControladorEscuela. Finalmente se considera importante destacar los métodos redefinidos de la súper clase Option, sin ellos no se podría utilizar componentes de JavaServerFaces, que en este caso nos permiten, como se verá más adelante, extraer un identificador y un nombre de los dropDownList, combos desplegables.



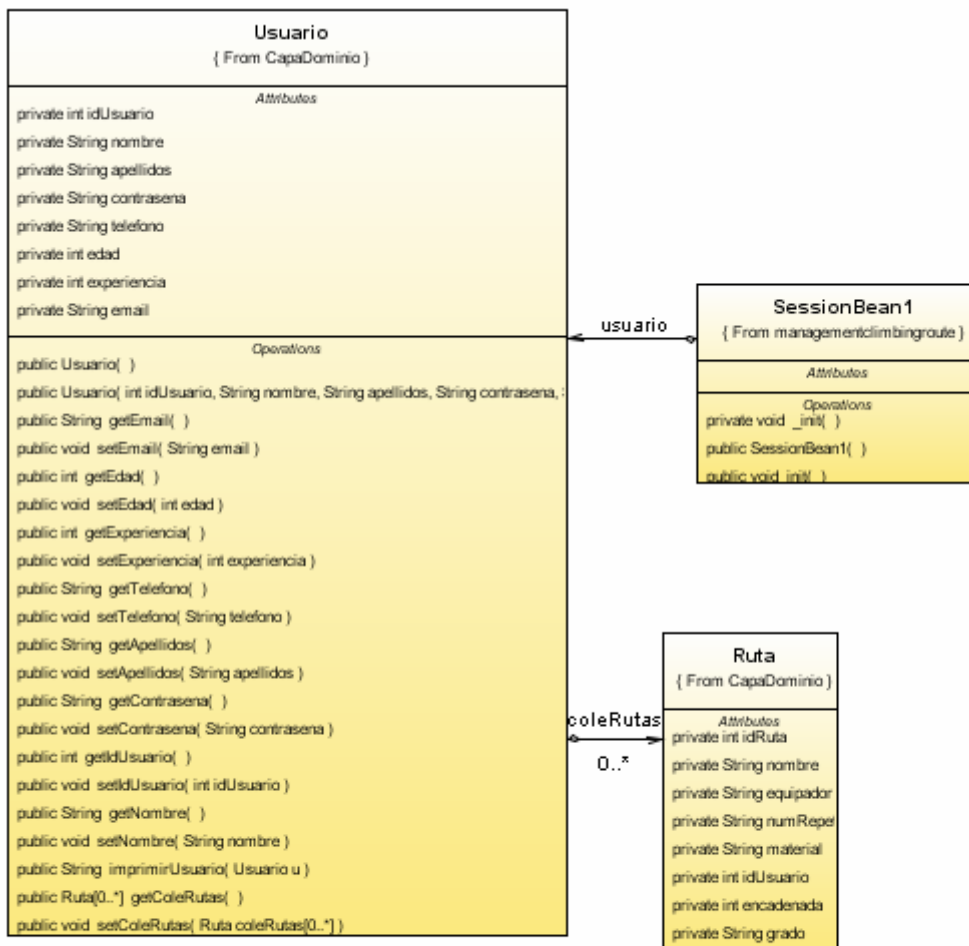
5.1.2 Diseño de la clase Sector

La clase Sector interactúa con la clase Ruta y tiene dependencia de la clase Escuela, no existen sectores sin Escuela, el controlador SessionBean posee diferentes atributos que hacen referencia a esta clase, como por ejemplo sector, objeto que contiene todos los atributos de clase Sector, sectores, una colección que contiene todos los sectores existentes, y coleSectores que posee todos los sectores que se pueden encontrar en una Escuela determinada. También es importante destacar que la clase Sector redefine métodos de la súper clase Option para obtener valores necesarios para la utilización de dropDownList, los combos despleglabes uno de los componentes de JavaServerFaces. En la clase Escuela cuando es necesario recuperar, modificar o insertar de se hace a partir de la SessionBean1, esta se comunica con el controlador respectivo de la clase, y transfiere o recuperar los datos necesarios a la persistencia.



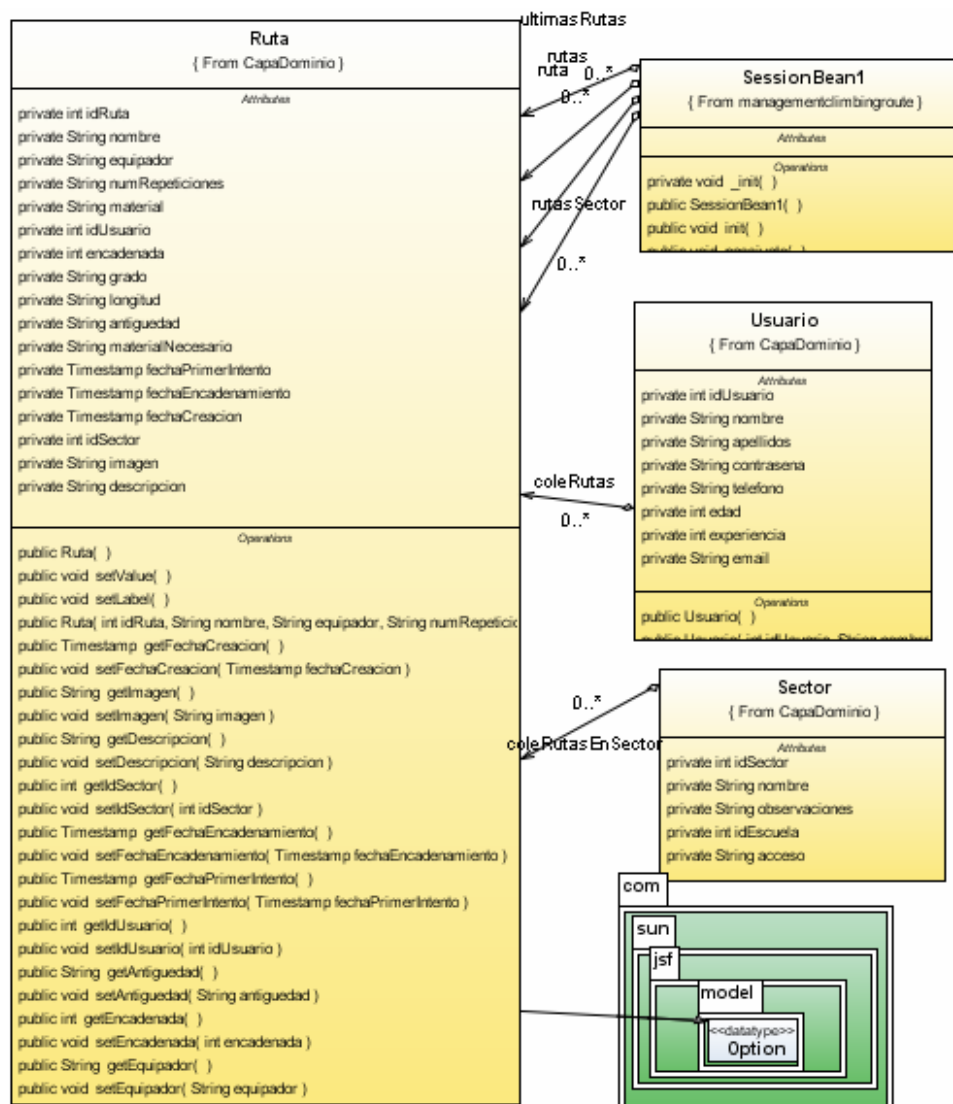
5.1.3 Diseño de la clase Usuario

Esta clase contiene los datos básicos para poder identificar al usuario, el atributo más interesante es la colección de rutas, cada Usuario poseerá una colección de rutas creadas, que podrá consultar y modificar. También podrá consultar rutas que no hayan sido introducidas por él, pero no formaran parte de su colección de rutas. También existe un atributo Usuario en la SessionBean1, para mostrar o modificar datos del Usuario, este recogerá los datos que provienen de la Persistencia, o bajará para modificar o insertar en la base de datos. Cualquier método de modificación, recuperación o inserción, partirá de la SessionBean1 y pasará por el controlador perteneciente a la clase Usuario.



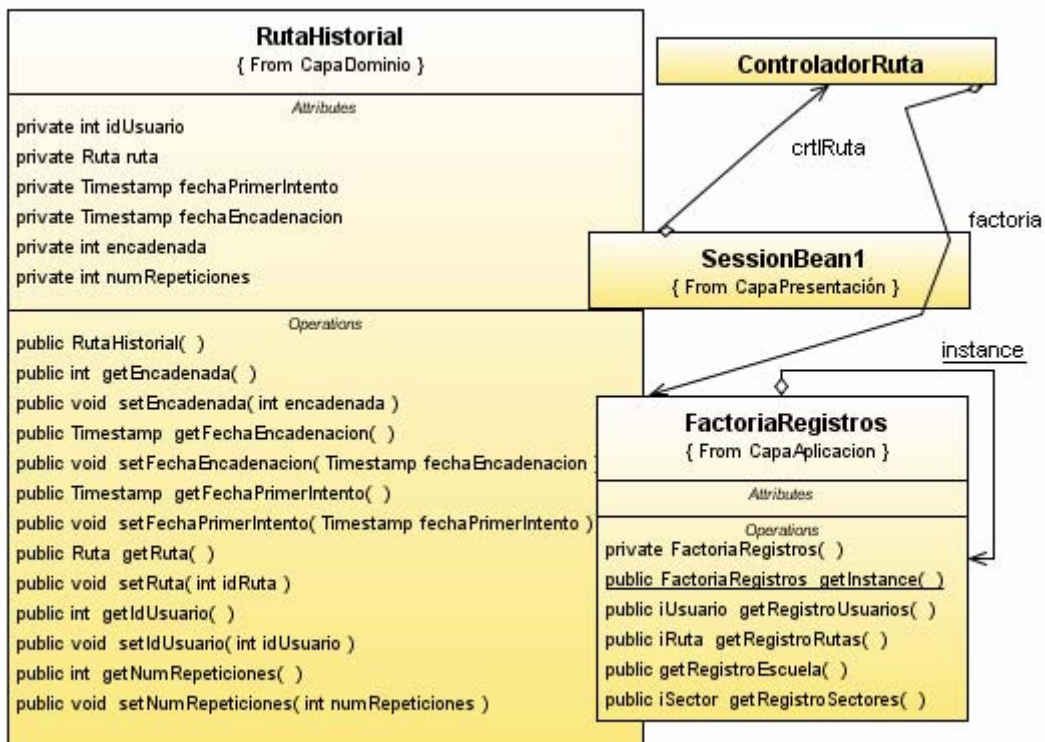
5.1.4 Diseño de la clase ruta

Esta es la clase más relevante del Dominio, sin las rutas no tendría sentido esta aplicación, una Ruta como hemos mencionado antes pertenece a un Sector, que a su vez pertenece a una Escuela, y un Usuario posee una colección de rutas. En la clase SessionBean1, tenemos diferentes atributos relacionados con Ruta, el atributo ruta, que nos valdrá para modificar, crear o eliminar una Ruta. También tenemos el atributo rutasSector, con el que podremos obtener información sobre las rutas específicas de un determinado Sector, el atributo rutas, que contendrá todas las rutas existentes en la base de datos. Y el atributo ultimasRutas, que contendrá las rutas más novedosas creadas por el usuario que las gestiona.



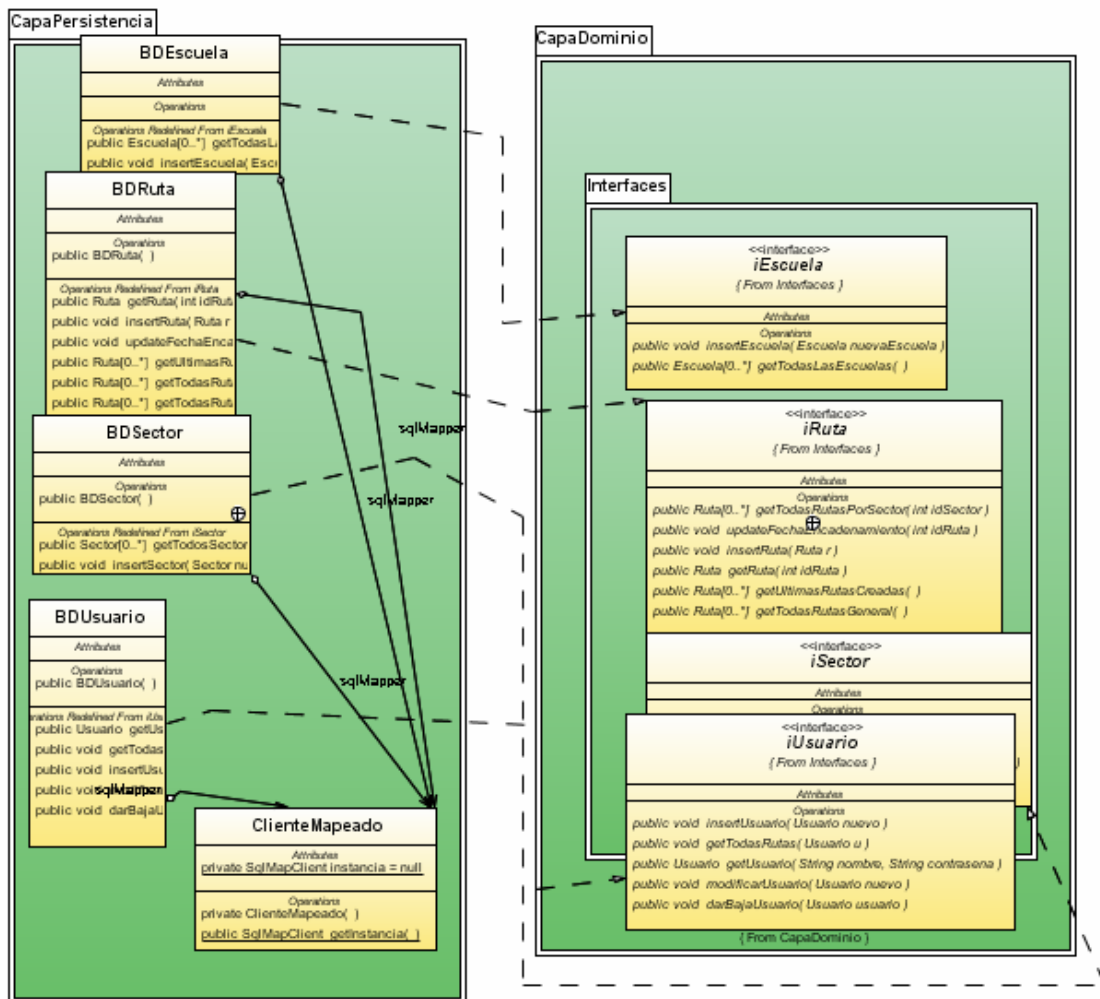
5.1.5 Diseño de la clase RutaHistorial

Esta clase fue la última en aparecer, básicamente esta clase gestiona el historial de rutas realizado por cada usuario. Un usuario cada vez que adjunta una nueva ruta a su historial crea un nuevo objeto de esta clase, que más tarde será insertado en el historial de rutas de la base de datos. Como datos a destacar es que comparte controlador con ruta, interface del dominio y archivo .xml de las inserciones, actualizaciones y eliminaciones que se producen en la base de datos mediante iBatis. Esta clase contiene un objeto ruta para facilitar la recuperación de datos de la ruta, y las modificaciones de las colecciones de la clase SesionBean.



5.2 Diseño de la capa persistencia

Esta capa o parte de la aplicación se crea para que no exista acoplamiento del resto de capas con la base de datos. Al crear esta capa conseguimos que la futura modificación de la misma no suponga un trauma para la aplicación. Veamos el diagrama.



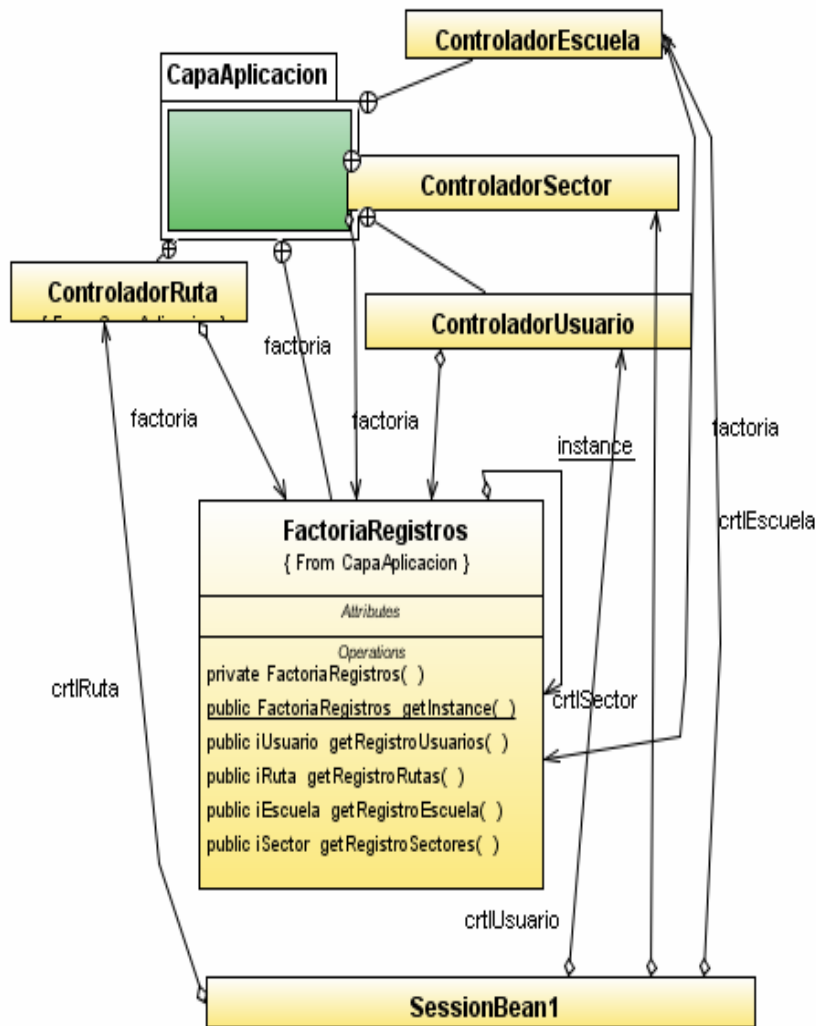
Todas las clases que forman parte de la capa Persistencia, poseen un atributo de la clase ClienteMapeado, llamado sqlMapper, este atributo es indispensable si queremos que los datos que se extraen de la BBDD se transformen en objetos del Dominio, esto es posible gracias a la librería iBatis. Este proceso se le llama mapeo de ficheros, que como mencionamos antes nos permite que la información recuperada pertenezca a objetos de la aplicación. Para que este proceso se realice con éxito también debemos tener en cuenta los ficheros .xml, que albergan las consultas a la BBDD, estos ficheros se encuentran en la CapaPersistencia.sql, habrá uno por cada objeto de la aplicación, no

es necesario la división, pero para que la aplicación no se transforme en una maraña caótica es una solución interesante. El fichero .xml a destacar dentro de la CapaPersistencia.sql es el SqlMapConfig.xml, que especifica el driver de conexión a la base de datos, el usuario y la contraseña, y los diferentes ficheros donde hallar las consultas.

5.3 Diseño de la capa aplicación

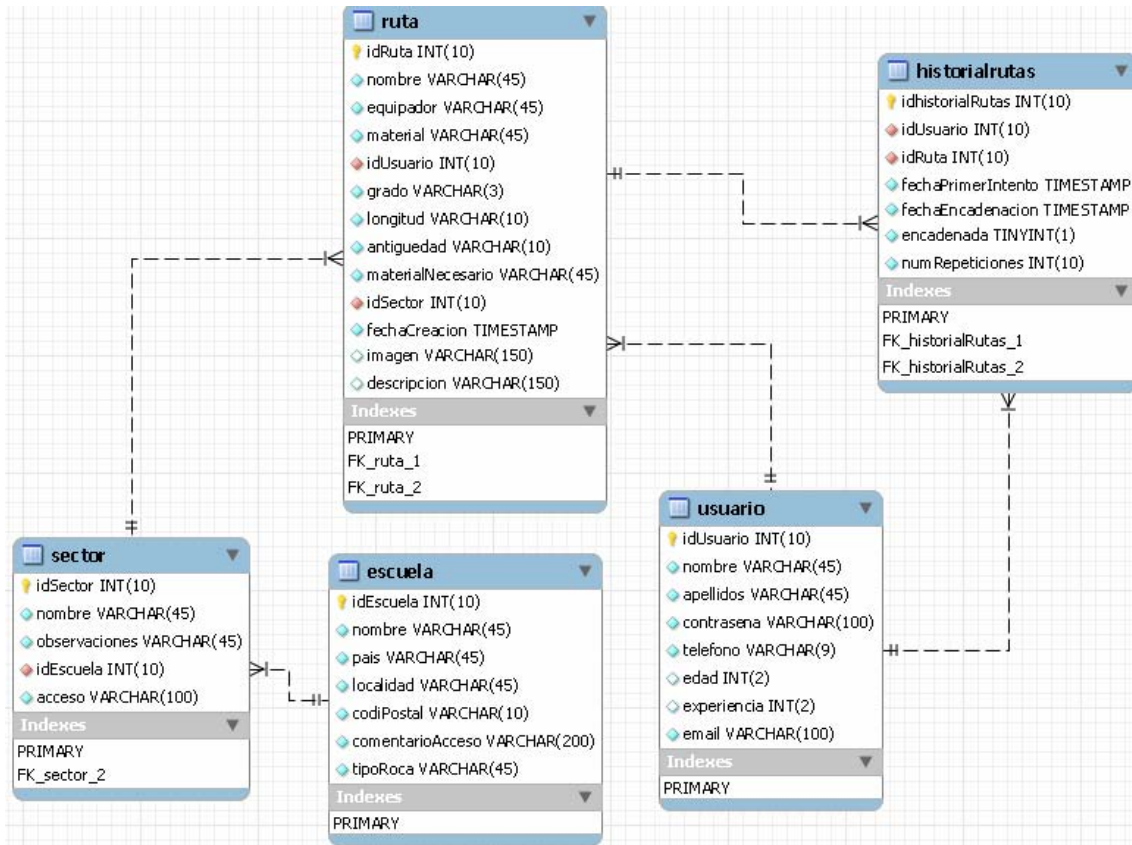
Esta capa contiene todos los controladores existentes en la aplicación, existen cuatro controladores uno para objeto del Dominio, exceptuando la clase historialRuta que comparte controlador con ruta, así que tenemos los siguientes controladores: ControladorRuta, ControladorEscuela, ControladorSector y ControladorUsuario. A parte existe la clase FactoriaRegistros.java, que será instanciada desde cada controlador, en esta clase se ha aplicado el patrón fachada que permite acceder a los datos sin tener que acoplarnos con la Persistencia, gracias a los métodos de la clase que nos devuelven instancias de las interfaces del Dominio. Cada uno de ellos contiene un atributo de clase que hace referencia a las interfaces, que a su vez permiten comunicar el Dominio con la Persistencia sin que exista acoplamiento. Por ejemplo el ControladorEscuela tendrá un atributo de clase del Tipo iEscuela con el que podrá acceder a los métodos contenidos en la clase que implementa iEscuela.java, en este caso BDEscuela.java, después de ser instanciado en la FactoriaResgistros.java.

También debemos tener en cuenta el Controlador, SessionBean1.java que hace referencia al Modelo Vista Controlador de JavaServerFaces, en esta aplicación nos permite conectar cualquier petición por parte de la Vista con el resto de la aplicación, es decir con los controladores referentes a los objetos del Dominio. Veamos el diagrama de clases de la capa aplicación con más detalle.



5.4 Diseño de la base de datos

La base de datos nos permite almacenar información, que posteriormente será recuperada, modificada o eliminada. Sin ella no podríamos tener constancia de los cambios que se producen, básicamente la aplicación no sería nada funcional sin una base de datos. El sistema gestor de base de datos utilizado es MySQL, sencillo, intuitivo y gratuito.



El modelo conceptual nos muestra las relaciones existentes entre las entidades que forman la base de datos, para interpretar correctamente la base de datos debemos tener en cuenta, las siguientes cosas; el símbolo amarillo en forma de llave, nos indica que es una clave primaria, los diamantes rojos nos indican que se considera una clave foránea, los diamantes azulados son datos que no pueden ser nulos y finalmente los diamantes blancos pueden ser nulos.

Las restricciones de la base de datos son:

- Un usuario puede poseer muchas rutas, en este caso seria el administrador o creador.
- Un sector puede poseer muchas rutas.
- Una escuela puede poseer muchos sectores.
- Un sector debe pertenecer una escuela.
- Una ruta pertenece a un usuario y un sector.
- El historialrutas pertenece a un usuario, a una ruta y un usuario puede tener muchos historiales.

6. Tecnologías utilizadas e implementación

En este proyecto se le ha dado especial importancia a la tecnología JavaServerFaces que también la hemos mencionado como el framework VisualWebJavaServerFace, y al framework iBatis que facilita la comunicación con la base de datos y el mapeo de objetos del dominio. Empezaremos explicando los que nos ofrece JavaServerFaces y acto seguido iBatis.

6.1 JavaServerFaces

JavaServerFaces tiene como objetivo el desarrollo de aplicaciones web, construyendo aplicaciones de una forma similar a como se realizan en aplicaciones de escritorio con Java Swing o cualquier otra API similar.

Anteriormente las aplicaciones Web se gestionaban mediante páginas JSP (Java Server Pages), que reciben las peticiones a través de formularios, y acto seguido se construyen páginas HTML (Hiper Text Markup Language), en estas páginas se utilizaban etiquetas con código java que permiten el acceso a base de datos, para mostrar cualquier tipo de dato, o bien realizar operaciones marginales de inserción, modificación o eliminación.

Con JavaServerFaces se nos facilita la construcción de esas páginas, mediante un entorno de trabajo, que gestionará las peticiones del usuario, traducidas a eventos que el servidor gestionará y volverá a mostrar al cliente, después de los cambios necesarios. Es importante destacar que la página será recargada en su totalidad, dato a tener en cuenta si la información de la página es muy extensa y la conexión que nos proporciona el proveedor es baja. Resumiendo en vez de devolver ventanas JFrame, JavaServerFaces nos proporciona una nueva página Html.

6.1.1 Características principales

A destacar que JavaServerFaces es un framework que posee interfaces de usuario en el servidor, y esta basado en el patrón MVC (Model Vista Controlador). Esto significa que tendremos unas clases.java, que recogerán los eventos que se produzcan en las páginas JSP, HTML y el servidor después de realizar los cambios pertinentes.

Los componentes más importantes de la tecnología JavaServerFaces son:

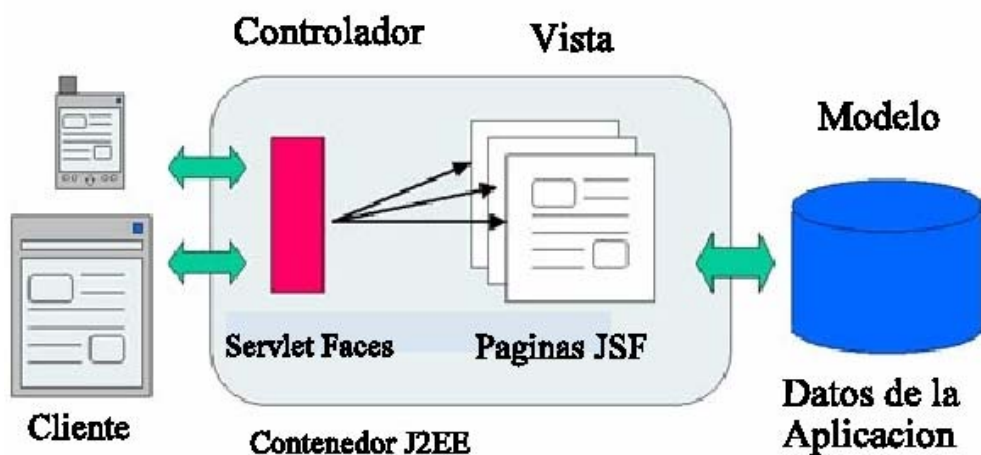
Una API y una implementación de referencia para:

- La representación de componentes de la interfaz de usuario y además poder manejar su estado.
- El manejo de eventos, validarlos en el lado del servidor y convertir datos.
- Poder definir la navegación entre las páginas de una forma bastante sencilla.
- Soportar internacionalización y accesibilidad.
- La librería de etiquetas JSP personalizadas para dibujar componentes dentro de JSP.

Gracias a lo mencionado anteriormente es relativamente sencillo conectar los eventos generados en el cliente con el servidor. Y podemos mapear los componentes de datos del usuario en un página de datos del servidor, y poder construir una interfaz de usuario con componentes extensible y reutilizables.

Modelo Vista Controlador en JavaServerFaces

Este patrón nos permite separar lógica de negocio(es decir cómo hacemos las cosas), de la lógica de control (qué cosas tenemos que hacer) y finalmente de la lógica de presentación (como relacionarnos con el usuario). Este patrón nos permite tener un punto de partida, un mantenimiento de la aplicación mas sostenible, basicamente la generación de un software estandarizado y normalizado.



Al utilizar este patrón también obtenemos una separación clara entre componentes, de esta forma podemos codificar por separado. Tendremos un API perfectamente definida y podremos modificar tanto la vista, como el controlador y el modelo con una dificultad moderada, y la conexión entre la vista y el modelo es dinámica, es decir se produce durante la ejecución no en durante la compilación.

6.1.2 Modelo

Llamamos modelo a ciertos datos que están relacionados con objetos de la realidad e intentamos representar en el dominio, es decir que modelan parte de la realidad sobre la que actúa la aplicación. El modelo trabaja sobre los datos de la aplicación. El modelo no tiene constancia ni de controladores, ni de vistas, por no tener no tiene ni referencias a ellos. En todo caso el sistema sería el encargado mantener los enlaces entre las vistas y el modelo. La parte del modelo referente al AltaUsuario, interactúa con JavaServerFaces mediante el archivo faces-config.xml, donde se identifica la clase AltaUsuario, donde se encuentra y si forma parte del ámbito de sesion o del request, veamos el código que nos facilita esa información y también el código referente a la sesion:

```
<managed-bean>
  <managed-bean-name>SessionBean1</managed-bean-name>
  <managed-bean-class>managementclimbingroute.SessionBean1</managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

```
<managed-bean>
  <managed-bean-name>AltaUsuarioFinal</managed-bean-name>
  <managed-bean-class>managementclimbingroute.AltUsuarioFinal</managed-bean-c
  <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

6.1.3 Vista

En la vista se representan los valores del modelo de forma visual, de esta manera el usuario puede ver datos referentes al modelo. La forma de interactuar con el modelo es mediante una referencia al mismo, esto se produce a través de las páginas JSP, por ejemplo:

```
<webuijsf:label binding="#{IndiceFinal.label2IdRuta}" id="label2IdRuta" text="
<webuijsf:hyperlink actionExpression="#{IndiceFinal.hyperlinkNombreRuta_action
    id="hyperlinkNombreRuta" text="#{ruta.nombre}" />
<webuijsf:label id="labelGrado" text="#{ruta.grado}" />
```

En la imagen superior, vemos un objeto que referencia al dominio, en este caso ruta, y como todo objeto tienes sus atributos, en este caso, ruta.nombre y ruta.grado, que nos mostraran en la pantalla el nombre y el grado de la ruta. Y de esta forma JavaServerFaces establece un vínculo entre la vista y el modelo, esto se produce gracias a la clase SesionBean, que contiene atributos relacionados con el modelo.

6.1.4 Controlador

El controlador se encarga de las peticiones que realiza el usuario, siempre trabajando con objetos que hacen referencia al modelo. Siempre que se produce un cambio en la vista el controlador se dispara para llevar a cabo la petición.

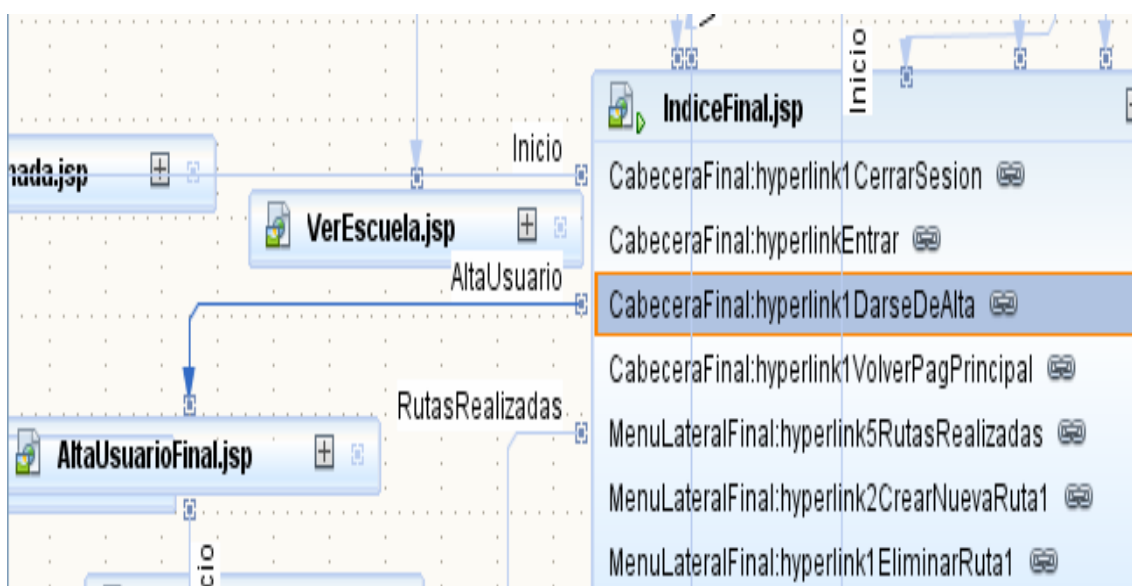
En cualquier aplicación el controlador se encarga de darle un significado a las peticiones del usuario, todo actuando sobre los datos representado en el modelo. El controlador aparece en escena cuando se realiza algún cambio en la vista o bien en la información del modelo. También hay que tener en cuenta que JavaServerFaces reaccionará a cualquier evento provocado por el usuario, procesándolo y realizando los cambios necesarios en el modelo a la vista. Veamos un ejemplo de código:


```
<webuijsf:hyperlink actionExpression="#{AltaUsuarioFinal.hyperlink1EnviarDatos_actionEnviarDatos
id="hyperlink1EnviarDatos"
style="color: rgb(0, 0, 0); font-size: 14px; left: 327px; top: 235px; position: absolut
text="Enviar datos"/>
```

En el trozo de código de la página altaUsuarioFinal.jsp el controlador recoge los datos y los envía al método AltaUsuarioFinal.hyperlinkEnviardatos_actionEnviardatos(), donde son tratados, y en este caso la clase SesionBean los distribuye a los métodos que se encuentran en la persistencia. El controlador de JavaServerFaces también se encarga de la navegación de las páginas, que esta definida en faces-config.xml, veamos un trozo de código para entender su funcionamiento.

```
<navigation-rule>
  <from-view-id>/IndiceFinal.jsp</from-view-id>
  <navigation-case>
    <from-outcome>AltaUsuario</from-outcome>
    <to-view-id>/AltaUsuarioFinal.jsp</to-view-id>
  </navigation-case>
```

Observando el código entenderemos que si queremos dar de alta a un usuario, desde la página IndiceFinal.jsp, tendremos que dirigirnos a AltaUsuarioFinal.jsp, AltaUsuario hace referencia la relación que une el link que se encuentra en IndiceFinal.jsp y AltaUsuarioFinal.jsp. Veámoslo de forma visual, que queda infinitamente más claro.



En JavaServerFaces también tenemos la opción de distribuir los links de forma Visual, aunque si el número de páginas es relativamente grande puede ser caótico.

6.1.5 Convertir y validar datos

Una vez que el usuario ha introducido un valor, el valor viaja del formulario hasta la clase java, si se produce una acción en el formulario, los valores se convierten y si fuera necesario se validan, veamos un ejemplo paso a paso de cómo hacerlo. Primero tenemos el campo que queremos, convertir y validar en este caso.



En la imagen superior vemos un label que contiene el nombre del campo, un input textfield, y el mensaje en rojo que esta vinculado al textfield, para vincular un el mensaje a textfield, debemos presionar sobre el mensaje ctrl+shift y dirigirlo hacia el textfield de esta forma quedarán vinculados. Seguidamente para poder trabajar con el textfield en el servidor, es decir en la clase.java que hace referencia a la página jsp, tenemos que hacer un binding al objeto textfield, hacer un binding no es más que crear un atributo en la clase java e instanciarlo para poder trabajar con él. Veamos como se hace, primero nos colocamos sobre el textfield, botón derecho y seleccionamos Add binding attribute, entonces nos aparecerá el siguiente código en la página jsp:

```

<webuijsf:textField binding="#{AltaUsuarioFinal.textField1ExperienciaAltaUsuario}"
id="textField1ExperienciaAltaUsuario" required="true" style="left: 96px; top: 1
  
```

Ahora ya poseemos el atributo en la clase java del servidor, instanciado y con sus métodos get y set.

```

public class AltaUsuarioFinal extends AbstractPageBean {

    private void _init() throws Exception {

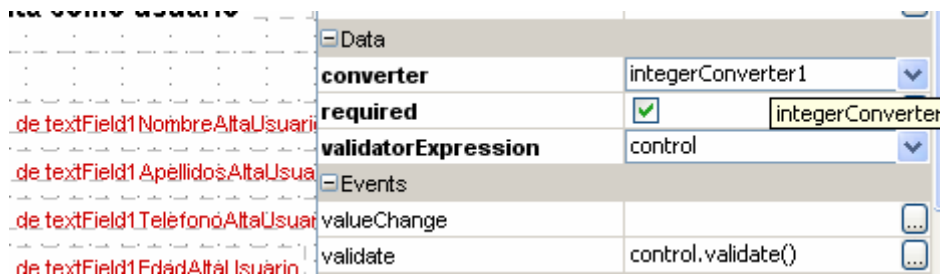
    }

    private TextField textField1NombreAltaUsuario = new TextField();
    private TextField textField1ApellidosAltaUsuario= new TextField();
    private TextField textField1TelefonoAltaUsuario= new TextField();
    private TextField textField1EdadAltaUsuario = new TextField();
    private TextField textField1ExperienciaAltaUsuario = new TextField();
    private TextField textField1EmailAltaUsuario = new TextField();
    private TextField textField1ContraseñaAltaUsuario = new TextField();

    public TextField getTextField1ApellidosAltaUsuario() {
        return textField1ApellidosAltaUsuario;
    }
}

```

Una vez hemos conseguido esto, podemos convertir el textfield en lo que deseemos en este caso en integer, para facilitar la inserción en la base de datos. Para convertir un campo de texto debemos acceder a las propiedades del mismo y modificar los siguientes campos:



En la imagen superior observamos la propiedad converter, en la que hemos seleccionado integerConverter, de esta forma cuando el valor del campo de texto llegue a la clase java ya será un integer. Veamos el cambio que se produce en el jsp:

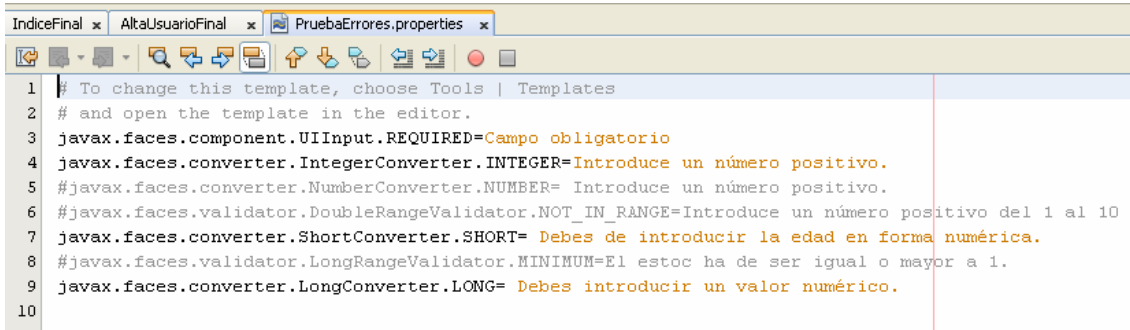
```

p: 120px; position: absolute" text="Experiencia"/>
taUsuario)" converter="#{AltaUsuarioFinal.integerConverter1}"
96px; top: 120px; position: absolute" validatorExpression="#{AltaUsuarioFinal.control.validate}"/>

```

En las propiedades también podemos observar la propiedad required, que obliga al campo de texto ha ser rellenado antes de enviar los datos a la clase .java, y gracias a esa propiedad, en el mensaje que hemos vinculado al campo de texto, se nos mostrará un mensaje de error sino rellenamos el campo antes ser enviado, bueno no podrá ser enviado sino rellenamos el campo de texto. Para definir el mensaje que queremos

mostrar cuando el campo de texto no este relleno tenemos que irnos a la clase, `bundle.properties`, que se suele encontrar en el paquete de configuración. Veamos que nos encontramos dentro de `bundle.properties`:



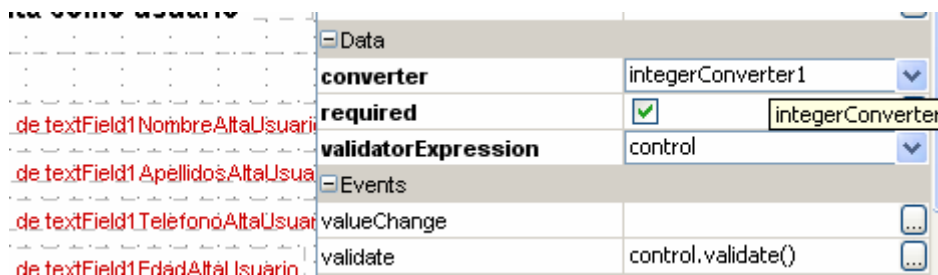
```

1  # To change this template, choose Tools | Templates
2  # and open the template in the editor.
3  javax.faces.component.UIInput.REQUIRED=Campo obligatorio
4  javax.faces.converter.IntegerConverter.INTEGER=Introduce un número positivo.
5  #javax.faces.converter.NumberConverter.NUMBER= Introduce un número positivo.
6  #javax.faces.validator.DoubleRangeValidator.NOT_IN_RANGE=Introduce un número positivo del 1 al 10
7  javax.faces.converter.ShortConverter.SHORT= Debes de introducir la edad en forma numérica.
8  #javax.faces.validator.LongRangeValidator.MINIMUM=El estoc ha de ser igual o mayor a 1.
9  javax.faces.converter.LongConverter.LONG= Debes introducir un valor numérico.
10

```

En ella observamos la propiedad `.REQUIRED` y el mensaje que se mostrará en caso de no ser relleno el campo.

A parte de este control, podemos crear nuestro propio control de errores `.Volvamos a ver la imagen de las propiedades del campo de texto:`



La propiedad `validatorExpression` y el evento `validate`, nos permiten controlar los errores como deseamos, en este caso hemos creado una clase `Control`, y el método `validate()`, que proviene de la clase `Validator` y hemos redefinido para adaptarlo a nuestras necesidades.

```

public class ControlErrores implements Validator{
    private String contraseña;
    private String contraseña2;

    public void validate(FacesContext context, UIComponent component, Object value) throws ValidatorExcepti
        if(component.getId().equals("textField1EmailAltaUsuario"))
        {
            if (!this.validaMail(value.toString())){
                throw new ValidatorException(new FacesMessage("Comprueba la dirección de correo (qqq@qqq.com)."));
            }
        }
    }
}

```

Además utilizando la clase FacesMessage, podemos definir el mensaje que esta vinculado al campo de texto. En este hemos validado el campo de texto como Integer pero existen más conversores como se observa la siguiente tabla.

TIPO DE CONVERSOR	
BigDecimalConverter	BigDecimal
BigIntegerConverter	BigInteger
NumberConverter	Number
IntegerConverter	Integer
ShortConverter	Short
ByteConverter	Byte
CharacterConverter	Character
FloatConverter	Float
DoubleConverter	Double
BooleanConverter	Boolean
DateTimeConverter	DateTime

6.1.6 Apache MyFaces

MyFaces forma parte de Apache, es un proyecto creado y mantenido por Apache, básicamente es una implementación que mejora las funcionalidades básicas de JavaServerFaces, a parte también posee diferentes funcionalades que pueden enriquecer JavaServerFaces las cuales son extendidas en los módulos de Tomahawk y Sandbox. En el caso particular de Tomahawk tiene como funciones específicas la agregación de controles, y Sandbox integra funcionalidades más específicas de javaScript y Ajax. Veamos con más detalle en el siguiente apartado la librería Tomahawk, que es la que nos ayudado a generar la paginación de las tablas de una forma relativamente automática.

Librería Tomahawk

Para poder utilizar la librería Tomahawk, es necesario agregar como librería al proyecto, una vez agregada, necesitamos incorporarla a las páginas .jsp donde la deseemos utilizar, para que sea reconocida tenemos que agregar la siguiente línea de código en la cabecera:

```
<jsp:root version="2.1" xmlns:f="http://java.sun.com/jsf/core" xmlns:h="http://java.s
xmlns:t="http://myfaces.apache.org/tomahawk" xmlns:webuijsf="http://www.sun.com/v
<jsp:directive.page contentType="text/html;charset=UTF-8" pageEncoding="UTF-8"/>
```

A partir de esta agregación ya podremos utilizar el tag “t” con sus diferentes propiedades. A destacar el tag t:datatable con el que podemos paginar y generar bucles de valores del modelo. Veamos un ejemplo:

```
<t:dataTable footerClass="standardTable Header" headerClass="standardTable Header" id="catalogoRutas"
reserveDataModel="false" rows="4" styleClass="scrollerTable" value="#{SessionBean1.ultimasRutas}" var="ruta">
<h:column>
webuijsf:imageHyperlink height="60" id="imageHyperlinkImagen" imageURL="#{ruta.imagen}" text="" width="60"/>
</h:column>
<h:column>
<table>
<tr>
<td>
<webuijsf:label binding="#{IndiceFinal.label1IdRuta}" id="label1IdRuta" text="#{ruta.idRu
<webuijsf:hyperlink actionExpression="#{IndiceFinal.hyperlinkNombreRuta_action}"
id="hyperlinkNombreRuta" text="#{ruta.nombre}" />
<webuijsf:label id="labelGrado" text="#{ruta.grado}" />
</td>
</tr>
<tr>
<td>
<webuijsf:label id="labelDescripcion" text="#{ruta.descripcion}" />
</td>
</tr>
```

En la imagen podemos observar el valor que coge, en este caso SessionBean1.ultimasRutas, que es una colección de rutas, hacemos el nombre más corto, le asignamos un alias, ruta y mostramos el nombre el grado y la descripción. También es interesante el hecho de que podamos definir el número de filas que queremos mostrar row= 4, y lo que observaremos en la siguiente imagen:

```

<t:dataScroller fastStep="10" for="catalogoEscuela" id="scroll_1" pageCountVar="pageCount" pageIndexVar="page
  paginator="true" paginatorActiveColumnStyle="font-weight:bold;" paginatorMaxPages="4"
  paginatorTableClass="paginator" styleClass="scroller">
  <f:facet name="first">
    <t:graphicImage border="1" url="/resources/primer.gif"/>
  </f:facet>
  <f:facet name="last">
    <t:graphicImage border="1" url="resources/ultimo.gif"/>
  </f:facet>
  <f:facet name="previous">
    <t:graphicImage border="1" url="resources/anterior.gif"/>
  </f:facet>
  <f:facet name="next">
    <t:graphicImage border="1" url="resources/siguiente.gif"/>
  </f:facet>
</t:dataScroller>
<t:dataScroller displayedRowsCountVar="displayedRowsCountVar" firstRowIndexVar="firstRowIndex" for="catalogoE
  id="scroll_2" lastRowIndexVar="lastRowIndex" pageCountVar="pageCount" pageIndexVar="pageIndex" rowsCountV
  <h:outputFormat styleClass="standard">
    <f:param value="#{rowsCount}"/>
    <f:param value="#{displayedRowsCountVar}"/>
    <f:param value="#{firstRowIndex}"/>
    <f:param value="#{lastRowIndex}"/>
    <f:param value="#{pageIndex}"/>
    <f:param value="#{pageCount}"/>
  </h:outputFormat>
</t:dataScroller>

```

El dataScroller que nos permite paginar la información, incluyendo imágenes para avanzar o volver atrás en la paginación.

6.2 iBatis

iBatis es marco de trabajo (framework) desarrollado por Apache Software Foundation, es de código abierto y está basado en capas, tiene la gran ventaja que puede ser

implementado en dos de los lenguajes de programación más extendidos en la actualidad, como .Net y Java. iBatis se ocupa de la capa de Persistencia, esta situado entre la lógica de negocio y la base de datos. La gran cualidad de iBatis es que asocia los objetos del modelo, con las sentencias SQL, esto se produce en fichero XML que contienen las consultas. iBatis facilita el mapeo de los objetos, simplificando la utilización de la base de datos. Es recomendable el uso de iBatis cuando queremos tener una curva de aprendizaje veloz, ya que al principio parece extraño y complicado, pero una vez has asimilado el funcionamiento te parece fascinante y sencillo, y como no de gran utilidad. También es interesante el hecho de que no requiera aprender un lenguaje de consultas, solo es necesario conocer el SQL. Pero no es recomendable su utilización en caso de desear una automatización completa y transparencia de la persistencia.

Podemos dividir la Persistencia entre tres capas:

- Capa Abstracción, es la interfaz que se comunica con la lógica de negocio, actuando a veces como patrón fachada.
- Framework de persistencia, es la interfaz que se comunica con el gestor de base de datos, gestiona los datos como un API.
- Capa Driver, que se encarga de la conexión con la base de datos utilizando un driver específico de la base de datos, en nuestro caso *com.mysql.jdbc.Driver*.

Para poder implementar iBatis, necesitamos estos componentes:

- DataMapper que proporciona el puente entre los objetos Java y la base de datos.
- Data Access Object, una abstracción que proporciona un API de acceso al resto de la aplicación y oculta la persistencia de objetos.

6.2.1 Implementación iBatis

En este apartado vamos a ver como se ha configurado iBatis en este proyecto, todo empieza una vez se ha adjuntado la librería iBatis al proyecto, una vez ya disponemos de ella, nos dirigimos a la capa persistencia, allí creamos una clase llamada

ClienteMapeado.java el nombre nos indica la una de las funciones más destacables de iBartis, mapear objetos. Veamos que contiene esa clase:

```
private static SqlMapClient instancia=null;
```

En la clase un atributo de la clase SqlMapConfig que nos permitirá instanciar iBatis y ponerlo en funcionamiento. Veamos como lo instanciamos:

```
String ruta="CapaPersistencia/sql/SqlMapConfig.xml";
Reader reader = Resources.getResourceAsReader(ruta);
instancia = SqlMapClientBuilder.buildSqlMapClient(reader);

reader.close();
```

Primero le comunicamos donde puede encontrar el fichero de configuración que le indicará donde se encuentran los diferentes ficheros.xml que nos permitirán trabajar con los objetos del dominio. En la clase SqlMapConfig.xml encontramos lo siguiente:

```
<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE sqlMapConfig
PUBLIC "-//ibatis.apache.org//DTD SQL Map Config 2.0//EN"
"http://ibatis.apache.org/dtd/sql-map-config-2.dtd">

<sqlMapConfig>

  <transactionManager type="JDBC" >
    <dataSource type="SIMPLE">
      <property value="com.mysql.jdbc.Driver" name="JDBC.Driver" />
      <property value="jdbc:mysql://localhost/bddclimbing" name="JDBC.ConnectionURL" />
      <property value="root" name="JDBC.Username" />
      <property value="root" name="JDBC.Password" />
    </dataSource>
  </transactionManager>

  <sqlMap resource="CapaPersistencia/sql/Usuario.xml" />
  <sqlMap resource="CapaPersistencia/sql/Ruta.xml" />
  <sqlMap resource="CapaPersistencia/sql/Sector.xml" />
  <sqlMap resource="CapaPersistencia/sql/Escuela.xml" />
  <sqlMap resource="CapaPersistencia/sql/RutaHistorial.xml" />

</sqlMapConfig>
```

Es muy importante intentar no modificar los no modificar en orden de los tags, y escribir correctamente todos los nombres de las clases xml, para intentar evitar cualquier error léxico que en el caso de que se produjese, es realmente difícil de detectar. En la primera parte del fichero observamos la versión del SqlmapConfig, acto

seguido en el tag <transactionManager> en tipo de conexión, el sistema gestor de base de datos que utilizamos en este caso MySQL, la base de datos a la que nos conectamos, el nombre del usuario y la contraseña. Y finalmente los archivos xml donde encontramos las diferente consultas a la base de datos, en este caso tenemos un fichero por objeto del dominio, teniendo en cuenta que tuta comparte el fichero con HistorialRuta.

Los Ficheros xml con las operaciones de la base datos los tenemos estructurados de la siguiente forma:

```
<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE sqlMap PUBLIC "-//ibatis.apache.org//DTD SQL Map 2.0//EN"
    "http://ibatis.apache.org/dtd/sql-map-2.dtd">

<sqlMap namespace="Usuario">

    <typeAlias alias="usuario" type="CapaDominio.Usuario"/>

    <select id="getUsuario" resultClass="usuario" parameterClass="usuario">
        select usuario.idUsuario,
            usuario.nombre,
            usuario.apellidos,
            usuario.contrasena,
            usuario.telefono,
            usuario.edad,
            usuario.experiencia,
            usuario.email
        from usuario
        where usuario.email=#email# and usuario.contrasena=#contrasena#
    </select>
```

El nombre del fichero se refleja en el namespace, después le facilitamos un alias, para disminuir el carro de texto, en este caso usuario, que hace referencia al CapaDominio.Usuario. una vez en este punto ya podemos efectuar las operaciones de relacionadas con la base de datos. Si queremos definir un select usamos el tag de la imagen, le damos un nombre al método, id=getUsuario, le decimos el tipo de objeto que nos tiene que devolver, resultClass= usuario, el alias definido anteriormente. El parameterClass= usuario, hace referencia al objeto que le llegara, un usuario. Es muy importante el orden de los atributos sea el mismo de la base de datos y a ser posible que

el nombre sea idéntico en el dominio y el base de datos, parece que no sea relevante pero en nuestro caso no provoco errores extraños y que generaban confusión. Otro dato a tener en cuenta es que las variables deben ir entre #x# de otra forma no conseguirá interpretarlas. Si en vez de un select queremos hacer un insert o un upadote solo tenemos que cambiar el tag, por insert o update, como se observa en los siguientes ejemplos.

```
<update id="modificarUsuario" parameterClass="usuario">
    UPDATE usuario SET
        usuario.nombre=#nombre#,
        usuario.apellidos=#apellidos#,
        usuario.telefono=#telefono#,
        usuario.edad=#edad#,
        usuario.experiencia=#experiencia#,
        usuario.email=#email#,
        usuario.contrasena=#contrasena#

    WHERE
        idUsuario=#idUsuario#
</update>
```

```
<insert id="insertSector" parameterClass="sector">
    insert into sector (
        nombre,
        observaciones,
        idEscuela,
        acceso
    )
    values(
        #nombre#, #observaciones#, #idEscuela#, #acceso#
    )
</insert>
```

Finalmente para utilizar todos estos métodos mostraremos la clase BDUsuario.java en la que llamamos a los métodos de los ficheros .xml.

```
usuario = (Usuario) sqlMapper.getInstancia().queryForObject("getUsuario", usuario);
```

Utilizamos el atributo sqlMapper, lo instanciamos y utilizamos el método de iBatis que nos devuelve un objeto usuario, especificado en el segundo parámetro del método, en el primero le comunicamos el id de método que tiene que seleccionar del fichero .xml y

que ejecutará la consulta. A destacar también el `.queryForObject` método que hace una consulta y devuelve un objeto.

```
ArrayList<Ruta> cole = new ArrayList();  
cole = (ArrayList<Ruta>) sqlMapper.getInstancia().queryForList("getTodasRutas", u.getIdUsuario  
u.setColeRutas(cole);
```

Este segundo ejemplo funciona de la misma manera, pero es bastante más increíble porque el `.queryForList` no devuelve una colección de datos en este caso del objeto ruta.

7. Conclusiones

La primera cosa que me viene a la cabeza a estas alturas del proyecto es todo el tiempo transcurrido y lo rápido que me ha pasado, y la cantidad de momentos en los que hubiese parado el tiempo para no tener esa sensación de descontrol, ansiedad y como no falta del susodicho. Pero todo eso parece que se ha quedado atrás y ahora empieza una nueva etapa.

Seguramente he cometido una infinidad de errores de planificación y estructuración de tiempo y he padecido más de lo necesario para realizar este proyecto, pero también es verdad que he estado a punto de abandonar la carrera más de una vez, no lo hice y ahora me enorgullezco de ello. Conozco muy bien mis limitaciones y conseguir finalizar esta carrera, me hace recapacitar sobre lo que he aprendido y que posiblemente aun puedo aprender muchas más. No soy un buen programador, no creo que lo llegue a ser nunca, demasiado esfuerzo y poca productividad en un mundo demasiado exigente, pero si se que soy capaz de esforzarme y conseguir mis objetivos. No se que voy hacer ahora, pero me gustaría dedicarme a algo que realmente se me de bien, mientras tanto seguiré trabajando como informático o programador y quiero dejar constancia de que eso es gracias a esta universidad y a todos los que he conocido aquí y me han enseñado. Todas estas palabras no tendrán ningún sentido para ustedes pero es importante para mí.

Volviendo a las conclusiones del proyecto, he descubierto una nueva tecnología como es JavaServerFaces, que realmente una vez empiezas a dominarla hace que la programación web sea bastante más sencilla y rápida, y te permite implementar los casos de uso de una forma muy parecida a la programación de escritorio. Sigo siendo un poco complicado leer los ficheros .jsp y difícil detectar los errores, al no poder seguir la trazada de debug, pero los componentes visuales te facilitan muchísimo el trabajo y la librerías de MyFaces, concretamente la Tomahawk ayudan mucho en este aspecto. Sinceramente me ha gustado y espero que continúe evolucionando para seguir disfrutando de ella. Aunque para mi la estrella de este proyecto es el marco de trabajo con la persistencia iBatis, hace la programación en la capa persistencia increíblemente fácil una vez entiendes su funcionamiento, al principio reconozco que acabe desquiciado, por la falta de información en los errores o la ausencia de ella, pero poco a poco y siendo muy metódico con el léxico descubres el gran trabajo que te ahora con el

mapeo de objetos. Menciono lo de ser metódico con el léxico porque errores incomprensibles para mi se solucionaron renombrando los atributos del dominio y la base de datos, para que fueran idénticos, parece ser que iBatis se confunde en caso de nombrarlos iguales. También es importante el orden de los mismos, para que no aparezcan errores extraños. Una vez aclarado esto me gustaría finalizar remarcando que espero continuar trabajando en esta aplicación aplicarle estilos, mejorar las funcionalidades, y ampliarla en medida de lo posible.

8. Bibliografía

<http://cafelojano.wordpress.com/2008/01/08/binding-un-arraylist-a-un-dropdownlist-de-jsf>

[http://209.85.229.132/search?q=cache:2xuFwLLwzREJ:www.di.uniovi.es/~dflanvin/home/%3Fdownload%3D06.%2520JSF%2520\(Java%2520Server%2520Faces\).pptx+combos+anidados+java+server+faces+JSF&cd=1&hl=es&ct=clnk&gl=es](http://209.85.229.132/search?q=cache:2xuFwLLwzREJ:www.di.uniovi.es/~dflanvin/home/%3Fdownload%3D06.%2520JSF%2520(Java%2520Server%2520Faces).pptx+combos+anidados+java+server+faces+JSF&cd=1&hl=es&ct=clnk&gl=es)

<http://www.netbeans.org/kb/60/web/intro.html>

<http://www.netbeans.org/kb/docs/web/fileupload.html>

http://www.javahispano.org/contenidos/es/comparativa_de_frameworks_web/

<http://www.scribd.com/doc/7545561/tutoria-jsf>

<http://weblogs.sqlteam.com/mladenp/archive/2008/10/21/Different-ways-how-to-escape-an-XML-string-in-C.aspx>

<http://www.netbeans.org/kb/60/web/helloweb.html>

<http://myfaces.apache.org/gettingstarted.html>

<http://myfaces.apache.org/download.html>

<http://www.apache.org/dyn/closer.cgi/myfaces/binaries/tomahawk-1.1.9-bin.zip>

<http://www.maestrosdelweb.com/editorial/internethis/>

<http://bocabit.com/tecnologia/informatica/internet/la-breve-historia-de-internet-2000-2009.php>

<http://es.wikipedia.org/wiki/Internet>

<http://www.adrformacion.com/cursos/intav/leccion1/tutorial2.html>

http://www.microsoft.com/spanish/msdn/netframework/framework20_InformacionCaract.msp

<http://java.sun.com/javaee/>

<http://www.sicuma.uma.es/sicuma/Formacion/documentacion/JSF.pdf>

http://es.wikipedia.org/wiki/Modelo_Vista_Controlador

<http://www.netbeans.org>

<http://ibatis.apache.org/javadownloads.cgi>

<http://wiki.netbeans.org/NavegandoPaginasVisualJSF>

<http://ibatis.apache.org/javadownloads.html>

<http://dev.mysql.com/downloads/mysql/5.1.html#downloads>

<http://luauf.com/2008/08/05/ejemplo-de-aplicacion-web-con-netbeans-y-visual-web-javascript-faces/>

Craig Larman, Uml y Patrones. Una introducción al análisis y diseño orientado y al proceso unificado. Prentice Hall, segunda edición 2003.

9. Glosario

DAO, Data Access Object: es un componente de software que suministra una interfaz común entre la aplicación y uno o más dispositivos de almacenamiento de datos, tales como una Base de datos o un archivo

iBatis: framework de código abierto basado en capas desarrollado por Apache Software Foundation, que se ocupa de la capa de Persistencia, se sitúa entre la lógica de Negocio y la capa de la Base de Datos.

Framework: En el desarrollo de software, es una estructura de soporte definida mediante la cual otro proyecto de software puede ser organizado y desarrollado.

Visual Web JSF: Nuevo Framework de Java mediante el cual se pueden generar páginas Web visualmente.

IDE, entorno de desarrollo integrado: Entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica GUI

MVC, Modelo Vista Controlador: es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.

MySQL: sistema de gestión de base de datos (SGBD) relacional, multihilo y multiusuario.

JSF, JavaServer Faces: JavaServer Faces (JSF): Framework para aplicaciones Java basadas en web que simplifica el desarrollo de interfaces de usuario en aplicaciones Java EE.

10. Anexo

10.1 Tutorial netBeans VisualWeb JavaServerFaces

El ejemplo consiste en una aplicación, una especie de pseudo-login (pues el usuario y password serán hardcoded), pero la idea central es mostrar el uso de Visual Web JavaServer Faces en NetBeans.

Para empezar, en mi ejemplo utilicé NetBeans 6.1, aunque no es excluyente pues en el ejemplo original del cual me basé lo hacía NetBeans 5.5. Lo importante reside en tener instalado el plugin de NetBeans Visual Web JSF, puedes ver si lo tienes instalado o instalarlo o actualizarlo desde el menú la opción Plugins del menú Tools.

Ahora bien, manos a la obra:

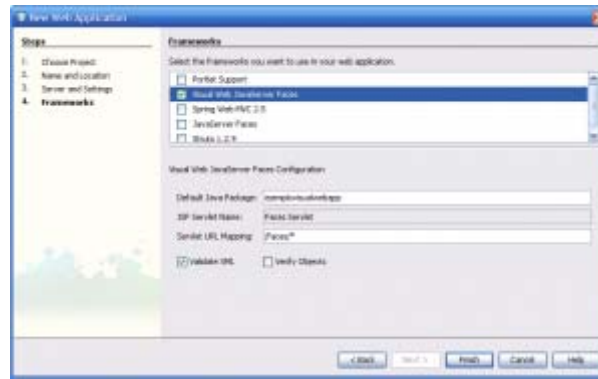
Una vez abierto NetBeans, seleccionar la opción New Project del menú File. En el primer diálogo que se abre, seleccionar de la categoría Web el tipo de proyecto Web Application.

En el siguiente paso, ingresar el nombre del proyecto y dar Next:



Luego, nos solicita el Server. En mi caso, por default tenía seleccionado Apache Tomcat 6.0.16, me limité en dar Next.

En el último paso nos solicita el framework a utilizar, tildamos Visual Web JavaServer Faces y finalizamos:

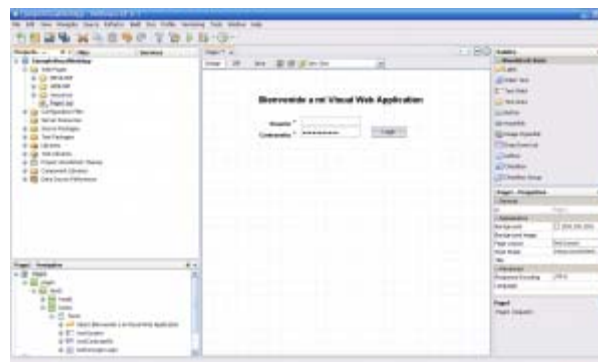


Finalizado el asistente, veremos que nos crea unas cuantas carpetas en la estructura del proyecto y por defecto una página Page1.jsp. En la misma, en tiempo de diseño insertamos los elementos de la paleta y modificamos algunas propiedades desde el cuadro de Properties:

Un **Label** con un título

Un **Text Field** y un **Password Field** con los **id** textUsuario y textContraseña. En la propiedad **label** de ambos podemos ingresar Usuario y Contraseña, para que queden bien rotulados. Además, para ambos, tildaremos la propiedad **required**.

Un **Button**, del cual modificamos las propiedades **id** (con el valor buttonLogin) y **text** (con el texto Login)

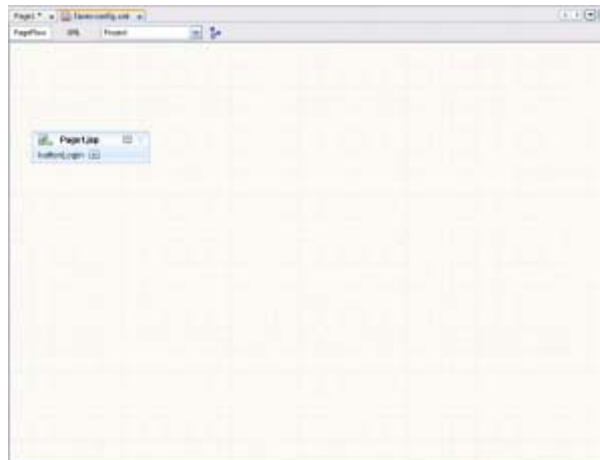


Para finalizar la edición de la página principal, haciendo clic con el botón derecho sobre cada Text Field seleccionamos del menú contextual la opción **Add Binding Attribute**, para poder más adelante acceder desde el código a tales objetos.

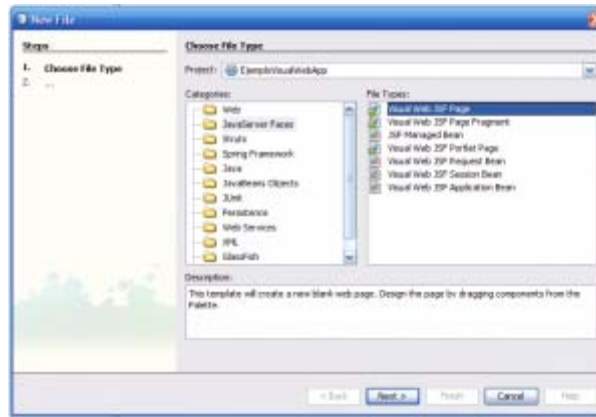
Luego, haciendo clic derecho sobre cualquier área vacía de Page1.jsp seleccionamos la opción **Page Navigation** del menú contextual:



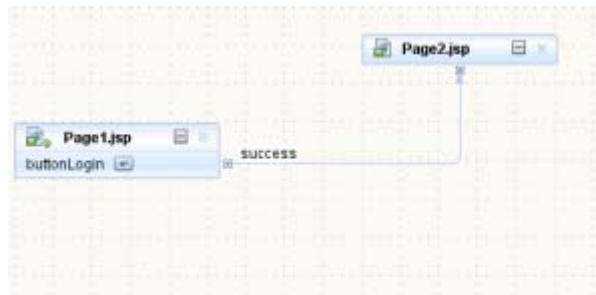
Inmediatamente se abrirá una nueva solapa **faces-config.xml** donde vemos ubicada una representación de la página con la que estuvimos trabajando, conteniendo la misma el objeto `buttonLogin`:



Haciendo clic derecho sobre el área de diseño, seleccionamos **New File** del menú contextual y luego de la categoría **JavaServer Faces, Visual Web JSF Page**:

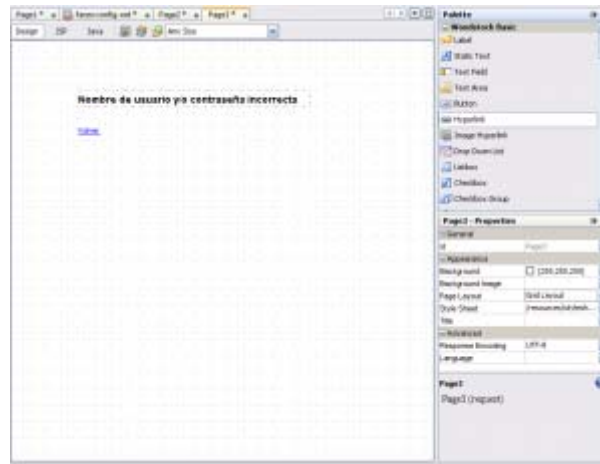


Ahora veremos que junto a Page1.jsp aparece Page2.jsp (si es que dejamos el nombre por defecto propuesto por el asistente). Haciendo clic sobre buttonLogin de Page1.jsp y manteniendo apretado el botón izquierdo del mouse trazamos una línea hasta Page2.jsp. Veremos que se crea un enlace entre ambas páginas, haciendo doble clic sobre en enlace, cambiamos el nombre por defecto del mismo (case1) por **success**:

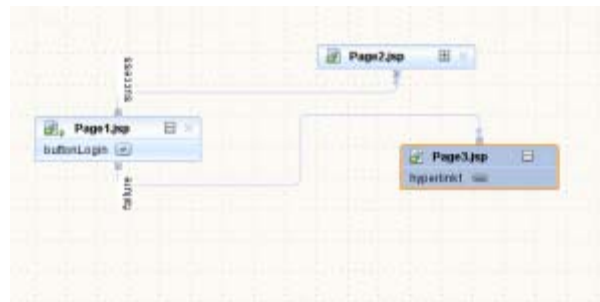


En Page2.jsp, insertamos un label con cualquier mensaje amistoso que indique que el login ha sido exitoso.

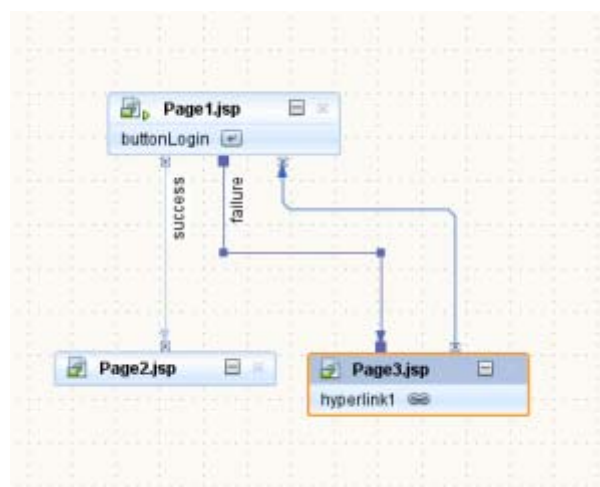
De la misma forma que creamos una página para el login exitoso (success) crearemos una nueva página (Page3.jsp) con un mensaje de "Nombre de usuario y/o contraseña incorrecta" y un **hyperlink** para volver:



Volviendo al esquema del faces-config.xml, creamos otro enlace desde el botón Login hacia Page3.jsp, cambiando el nombre por defecto (case1) por **failure**.



Para finalizar la configuración del esquema de navegación de páginas, creamos un nuevo enlace desde el elemento hyperlink1 (si es que no le cambiamos el id por defecto) de Page3.jsp con Page1.jsp. Además, si queremos podemos reorganizar un poco la distribución de las páginas:

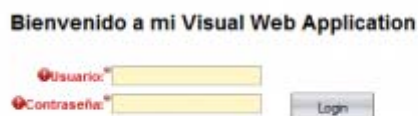


Volviendo al diseño de la página de ingreso (Page1.jsp) hacemos doble clic sobre buttonLogin y escribimos la porción de código que valide usuario y contraseña. Como dijimos en un principio, los valores para los mismos estarán en memoria.

```
public String buttonLogin_action() {
    if (textUsuario.getText().equals("demo") &&
textContraseña.getText().equals("demo")) {
        return "success";
    } else {
        return "failure";
    }
}
```

Para finalizar solo queda correr la aplicación, seleccionando la opción **Run Main Project** del menú **Run**, o simplemente presionando F6. Esto puede demorar un tiempo, dependiendo de tu equipo, pues NetBeans inicia el servidor de aplicaciones (Apache Tomcat en mi caso), hace el deployment de la aplicación y finalmente abre nuestro navegador por defecto con la página que diseñamos.

Con el navegador abierto y la página de login en frente, para probar de qué sirvió setear la propiedad required del Text Field y el Password Field apretamos directamente el botón Login. Veremos como nos marca en rojo los campos requeridos y nos lo vuelve a solicitar:



Luego, ingresaremos usuario y contraseña, pero equivocando uno o ambos valores:

Nombre de usuario y/o contraseña incorrecta

[Volver](#)

Finalmente, ingresamos usuario y contraseña correcta (demo demo):

Bienvenido !!!

Con estos pasos, no deberías tener problemas, pues el ejemplo publicado no es más que el paso a paso de lo que yo hice en mi PC de desarrollo.

10.2 Como conectar la PDA al servidor apache

Para que poder acceder desde el PDA al PC debes hacer lo siguiente:

1. Conectar el PDA a través de la misma red (WiFi)
2. Consultar la dirección IP del PC
3. Desde el PDA intentar abrir: http://ip_del_pc:8080/proyecto/capturar.jsp

10.3 Contenido de los CD's

Artículo referente al proyecto.

Aplicación ManegementClimbingRoute.

NetBeans 6.5 con componentes VisualWebJavaServerFaces, incluye servido apache.

mysql-connector-java-5.1.7.

mysql-workbench-oss-5.1.18a-win32.

mysql-gui-tools-5.0-r17-win32.

mysql-6.0.9-alpha-win32.(Servidor).

ibatis-2.3.4.726.jar

tomahawk-1.1.9-bin

