UOC

TFC JEE

Gestión académica de un centro de formación

Sergio Britos Arévalo
ETIS

José Juan Rodríguez

14/01/2009

RESUMEN

Este proyecto fin de carrera tiene como objetivo presentar la tecnología Java EE como un entorno viable en el que realizar aplicaciones WEB distribuidas.

Para la realización del proyecto he seguido el método en cascada tradicional: análisis, diseño e implementación.

En este documento describiré el uso de patrones y frameworks empleados para la implementación del producto final.

Finalmente, se desarrolla la implementación el producto Gestión académica de un centro de formación.

PALABRAS CLAVE

Java
JEE
SEAM Framework
Java Server Faces
Jboss Application Server
Gestión académica
Facelets
EJB
Transacción
Autentificación
Autorización

Índice del contenido

INTRODUCCIÓN

Justificación del TFC. Punto de partida y aportaciones

La tecnología Java Enterprise Edition (JEE de aquí en adelante) es una tecnología que lleva años en el mercado y con la que se han realizado multitud de aplicaciones empresariales. Sin embargo, los detractores de esta tecnología siempre le han achacado un nivel de dificultad demasiado elevado para el desarrollo de aplicaciones. Todo esto cambió radicalmente con la aparición de JEE 5, donde se recomienda el uso de POJO's (Plain Old Java Objects) para el desarrollo de aplicaciones empresariales en Java, cosa que simplifica enormemente el desarrollo de dichas aplicaciones.

En estos últimos años, también han tomado auge conceptos como: Ajax, Web 2.0 y RIA (Rich Internet Applications). Esto no es más que intentar que el navegador pueda ejecutar aplicaciones WEB que se asemejen en cuanto a nivel de interacción a las aplicaciones de escritorio clásicas.

Durante estos últimos años también se está observando como proyectos Open Source están teniendo éxito.

En este TFC, se verá como es posible desarrollar aplicaciones JEE de una manera sencilla (si lo comparamos con versiones anteriores a la 5), dotándolas de capacidades AJAX. Todo ello haciendo uso de herramientas Open Source. De hecho, para la confección de la casi totalidad de este TFC se han empleado herramientas Open Source.

Para llevar a cabo el TFC, se desarrollará la *Gestión Académica de un centro de formación* siguiendo la metodología de desarrollo clásica.

He apostado por la tecnología JEE, debido a que es un estándar en auge y con gran demanda en el entorno laboral. Además, muchos frameworks y herramientas están construidos para esta tecnología. Con toda seguridad que existen otros frameworks con la misma potencia o superior incluso, pero el soporte que el mercado da a una tecnología es clave para su elección.

A nivel personal, este TFC me permitirá introducirme en el desarrollo WEB y empresarial con la tecnología JEE. De hecho, gracias a este TFC he podido implantar mi primera aplicación JEE en la empresa en la cual trabajo.

Objetivos

La realización de este proyecto me permitirá poner en práctica todos los conocimientos adquiridos durante los estudios en la UOC en cuanto a orientación a objetos, Java, así como el ciclo de vida clásico para el desarrollo de aplicaciones.

Para realizar la implementación de este proyecto emplearé la tecnología JEE, cosa que me permitirá adquirir unos conocimientos básicos de dicha tecnología.

También me permitirá adquirir conocimientos de un Framework basado en JEE que está en auge y, que de hecho, se ha elegido como modelo para la siguiente especificación de JEE (JEE 6). Es el framework Open Source Jboss SEAM, creado por Jboss MiddelWare.

Otros de los objetivos que me he propuesto es el uso de herramientas Open Source para la realización del proyecto. En este proyecto intentaré demostrar como con el uso de herramientas Open Source es posible desarrollar aplicaciones del mundo real.

Para poner en práctica todo ello, desarrollaré una aplicación basada en la *gestión académica de un centro de formación.* Esta aplicación tendrá como principal objetivo la automatización de ciertas tareas propias de un centro educativo como pueden ser.

- Gestión de alumnos
- Gestión de la matrícula
- Gestión del expediente académico
- Gestión de cursos
- Gestión de períodos

Enfoque y método seguido

El método seguido para la elaboración del proyecto lo podríamos dividir en dos partes: la primera teórica y la segunda totalmente práctica.

En cuanto a la parte teórica comenzamos entregando un plan de trabajo que consistía en exponer un proyecto a realizar y la planificación temporal de dicho proyecto.

La parte teórica continúo con la entrega del análisis y diseño de la aplicación.

Mientras estaba desarrollando la parte teórica, fui recopilando información sobre la tecnología JEE y frameworks que me podrían ser útiles a la hora de desarrollar la aplicación final.

Una vez entregado el análisis y diseño, comencé con la elaboración del producto final.

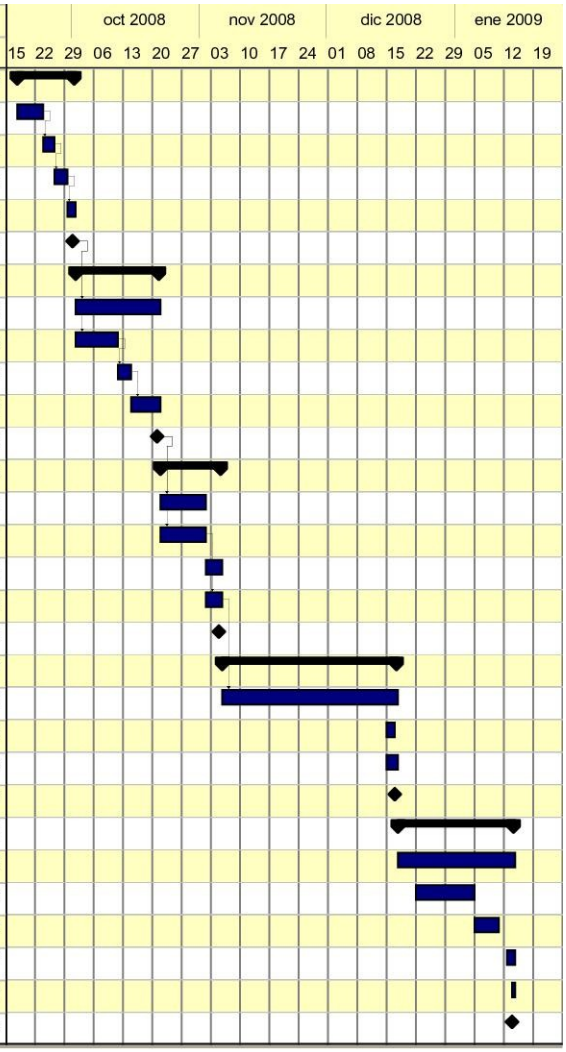
Finalmente, redacté este documento (la memoria) y una pequeña presentación virtual sobre el proyecto realizado.

Planificación y diagrama Gantt

La tabla siguiente muestra la distribución de las diferentes tareas. Como se puede observar en algunas de las fases he reservado el tiempo que creo necesario para investigar y practicar con las diferentes tecnologías a utilizar.

Tarea	*Previsión Días*	*Fecha Inicio*	*Fecha Fin*
Fase 1. Análisis previa	**15**	**18/09/2008**	**01/10/2008**
Elaboración análisis y documentación	6	18/09/2008	23/09/2008
Creación plan de trabajo	3	24/09/208	26/09/208
Estudio tecnología a utilizar	3	27/09/2008	29/09/2008
Revisión Plan de Trabajo	3	30/09/2008	01/10/2008
Entrega PAC1	0	01/10/2008	01/10/2008
Fase 2. Análisis de requerimientos	**20**	**02/10/2008**	**21/10/2008**
Estudio tecnología a utilizar	20	02/10/2008	21/10/2008
Elaboración análisis de requerimientos	10	02/10/2008	11/10/2008
Creación del prototipo	3	12/10/2008	14/10/2008
Crear documento de análisis	5	15/10/2008	19/10/2008
Fase 3. Diseño	**15**	**22/10/2008**	**05/11/2008**
Estudio tecnología a utilizar	11	22/10/2008	01/11/2008
Elaboración del diseño	11	22/10/2008	01/11/2008
Montar entorno de trabajo	4	02/11/2008	05/11/2008
Crear documento de diseño	4	02/11/2008	05/11/2008
Entrega PAC2	0	05/11/2008	05/11/2008
Fase 4. Implementación parcial	**42**	**06/11/2008**	**17/12/2008**
Codificación	40	06/11/2008	17/12/2008
Elaboración documento implementación	2	15/12/2008	16/12/2008
Crear EAR del proyecto	2	15/12/2008	17/12/2008
Entrega PAC3	0	17/12/2008	17/12/2008
Memoria y producto final	**28**	**18/12/2008**	**14/01/2008**
Terminar codificación (producto final)	26	18/12/2008	14/01/2009
Elaboración de la memoria	14	22/12/2008	04/01/2009
Elaboración de la presentación	6	05/01/2009	10/01/2009
Test funcional completo del proyecto	2	13/01/2009	14/01/2009
Crear EAR del proyecto	1	14/01/2009	14/01/2009
Entrega memoria y producto final	0	14/01/2009	14/01/2009

Name	Start	Finish
Fase 1: Análisis previa	**18/09/2008**	**01/10/2008**
Elaboración análisis y documentación	18/09/2008	23/09/2008
Creación plan de trabajo	24/09/2008	26/09/2008
Estudio tecnología a utilizar	27/09/2008	29/09/2008
Revisión Plan de Trabajo	30/09/2008	01/10/2008
Entrega PAC 1	01/10/2008	01/10/2008
Fase 2: Análisis de requerimientos	**02/10/2008**	**21/10/2008**
Estudio tecnología a utilizar	02/10/2008	21/10/2008
Elaboración análisis de requerimiento	02/10/2008	11/10/2008
Creación del prototipo	12/10/2008	14/10/2008
Crear documento de análisis	15/10/2008	21/10/2008
Fin del análisis de requerimientos	21/10/2008	21/10/2008
Fase 3: Diseño	**22/10/2008**	**05/11/2008**
Estudio tecnología a utilizar	22/10/2008	01/11/2008
Elaboración del diseño	22/10/2008	01/11/2008
Montar entorno de trabajo	02/11/2008	05/11/2008
Crear documento de diseño	02/11/2008	05/11/2008
Entrega PAC 2	05/11/2008	05/11/2008
Fase 4: Implementación Parcial	**06/11/2008**	**17/12/2008**
Codificación	06/11/2008	17/12/2008
Elaboración documento implementaci	15/12/2008	16/12/2008
Crear EAR del proyecto	15/12/2008	17/12/2008
Entrega PAC 3	17/12/2008	17/12/2008
Memoria y producto final	**18/12/2008**	**14/01/2009**
Terminar codificación (Producto final)	18/12/2008	14/01/2009
Elaboración de la memoria	22/12/2008	04/01/2009
Elaboración de la presentación	05/01/2009	10/01/2009
Test funcional completo del proyecto	13/01/2009	14/01/2009
Crear EAR del producto final	14/01/2009	14/01/2009
Entrega memoria y producto final	14/01/2009	14/01/2009



Productos obtenidos

Los productos obtenidos en este proyecto son:

- Memoria técnica. Recoge todo lo relacionado con la tarea de investigación necesaria para poder realizar el TFC.
- Software desarrollado.
  - Zips necesarios para el proyecto
  - BD para el proyecto
- Presentación virtual. Es una pequeña presentación que muestra un pequeño resumen o síntesis de los puntos claves en el desarrollo del TFC.

Organización de la memoria

El resto de capítulos de la memoria describe el proceso de empleado para llegar a la implementación del producto final.

Para ello he empleado el ciclo clásico de desarrollo de software: análisis, diseño e implementación del producto final.

Los dos capítulos que siguen a continuación detallan específicamente la fase de análisis de requerimientos y el diseño de la aplicación final.

ARQUITECTURA DEL PROYECTO

Introducción

El proyecto "gestión académica de un centro de formación" tiene como principal objetivo la automatización de ciertas tareas propias de una academia como pueden ser.

- Gestión de alumnos
- Gestión de la matrícula
- Gestión del expediente académico
- Gestión de cursos

Descripción de las funcionalidades

Componentes del proyecto

He dividido la aplicación en una serie de módulos, cada uno de los cuales con una autonomía propia.

El módulo *Gestión de alumnos* será el encargado del mantenimiento de alumnos en el sistema (CRUD de alumnos). Es decir, permitirá dar de alta, baja, modificar y visualizar los datos de un alumno. El **secretario/a** será el encargado de gestionar este módulo.

El módulo *Gestión de asignaturas* permitirá al **docente** realizar el mantenimiento de las diferentes asignaturas. Principalmente dará de alta asignaturas, les asignará los créditos correspondientes y cuando una asignatura deje de ofertarse, la dará de baja.

Otro de los módulos que mantendrá el **docente** será la *Gestión de Cursos*. El docente creará los cursos y les asignará las asignaturas correspondientes.

El módulo *Gestión de la matrícula* permitirá al **secretario/a** realizar la matrícula de alumnos. Los alumnos aportaran toda la documentación necesaria una vez validada por la secretaria, realizará la matrícula. El proceso de validación de la documentación será un proceso totalmente manual. No habrá ningún automatismo que facilite esta tarea.

El módulo *Gestión del expediente académico* permitirá al **profesor** la entrada de las calificaciones de sus alumnos. Por otro lado, permitirá a un **alumno** la consulta de sus calificaciones. Estos últimos podrán pedir la revisión de la calificación si no están de acuerdo con la misma.

Finalmente, tenemos la *Gestión de informes*. En particular habrán dos informes:

- Calificaciones por asignatura. Es un informe que gestionará el **profesor**. El profesor seleccionará una asignatura* y aparecerán todas las calificaciones de los alumnos de esa asignatura. Este informe se podrá generar desde la opción Entrada de calificaciones por lo que una vez haya finalizado el periodo de entrada de calificaciones del periodo actual, ya no se podrá generar.
- Boletín para el alumno. Son las calificaciones de los alumnos. Lo podrá generar el **alumno** desde la opción *Ver Calificaciones*.

NOTA

- Esta primera versión de la aplicación no gestionará la relación de un profesor con las asignaturas que imparte. En el caso de las calificaciones por asignatura y la entrada de calificaciones, el profesor verá una lista con todas las asignaturas.

Descripción del funcionamiento

El esquema típico de funcionamiento de la aplicación es el siguiente:

- Un **consejo docente** se reúne y discute las asignaturas que se ofrecerán en el centro de estudios. También decidirá dar de baja asignaturas que se han impartido, pero que por diversas razones ya no se continuarán impartiendo.
- El mismo **consejo docente** decide como agrupar una serie de asignaturas en cursos. Por lo general, los cursos tendrán una carga lectiva semestral.
- Dos veces al año se abre el periodo de matriculación. La **secretaria** se encarga de toda la gestión de alumnos y de validar que cada uno de los alumnos cumple con los requerimientos necesarios para matricularse en un curso. Una vez comienza el curso, el periodo de matriculación queda cerrado.
- Una vez finalizados los exámenes, el **profesor** entra las calificaciones en el sistema. Hay que tener en cuenta que en la primera versión de la aplicación, la aplicación no gestionará la vinculación de asignaturas a profesores.
- El **alumno** entra en el sistema, y puede consultar sus calificaciones. Primero selecciona un periodo y si es un periodo abierto para revisión, puede pedir la revisión de las calificaciones. Si es un periodo cerrado sólo podrá consultar sus calificaciones históricas. Opcionalmente puede generar un PDF con sus calificaciones.
- El **profesor**, informado por la secretaria puede volver a modificar la calificación del alumno.
- El **alumno** cuando vuelva a entrar en el sistema, podrá comprobar el

estado de su alegación (rechazada o aceptada). En ningún caso podrá pedir más de una revisión.

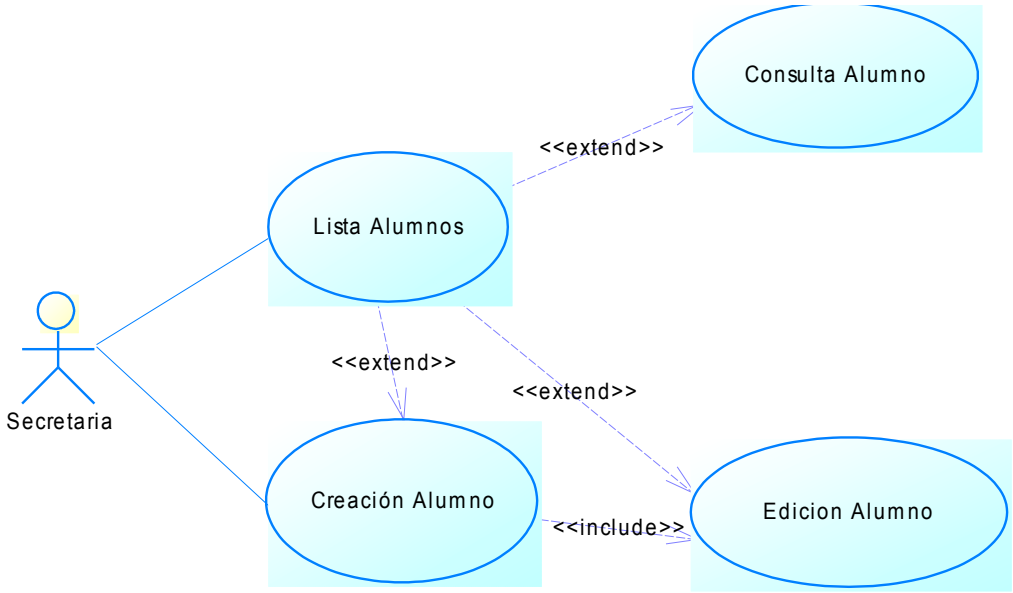
- Por último, el periodo docente finaliza y al cabo de unos días se inicia un nuevo periodo de matriculación. El cierre del periodo docente lo llevará a cargo la **secretaría** con una opción de programa.

Análisis de requisitos

Una de las fases del desarrollo del TFC fue la fase de análisis. En esta fase, se establecieron los límites de funcionalidad que tendría el producto final. Además documenté en forma de diagramas de secuencia, el flujo entre pantallas. Finalmente, presenté un prototipo de pantallas de la aplicación.

Por falta de espacio, no se hará una descripción detallada de cada uno de los casos de uso. Se detallarán los casos de uso más complejos y un ejemplo de CRUD (mantenimiento). El resto siguen prácticamente el mismo patrón y sólo los enumeraré.

Módulo gestión de alumnos



Caso de uso número 1: "Lista Alumnos"	
Resumen de la funcionalidad	Muestra una lista con los datos de los alumnos
Papel dentro del trabajo del usuario	Consulta de alumnos por diferentes criterios de selección. Usado frecuentemente en periodo de matriculación
Actores	Secretaria
Precondición	Ninguna, es una consulta
Flujo normal	1. La secretaria entra en la opción Gestión de alumnos 2. Aparece una lista con todos los alumnos ordenada por apellidos y nombre del alumno 3. La secretaria puede desplazarse por la lista y hacer scroll para ver todos los alumnos. 4. La secretaria hace clic en el botón salir.
Flujo alternativo	3. La secretaria hace clic en el botón *Consultar* para ver los datos detallados de un alumno (ver caso de uso "Consulta Alumno") 3. La secretaria hace clic en el botón *Editar* para editar los datos del alumno seleccionado (ver caso de uso "Edición alumno"). 3. La secretaria hace clic en "Nuevo Alumno" para *Crear* un nuevo alumno (ver caso de uso "Nuevo Alumno") 3. Opcionalmente la secretaria puede filtrar por nombre, apellidos, dni y población de nacimiento del alumno.
Postcondición	Ninguna, es una consulta
Caso de uso número 2: "Nuevo Alumno"	
Resumen de la funcionalidad	Crea un nuevo alumno en el sistema
Papel dentro del trabajo del usuario	Caso de uso muy frecuente en periodo de matriculación
Actores	Secretaria
Casos de uso relacionados	Edición alumno
Precondición	El alumno a crear no existe en el sistema
Flujo normal	1. La secretaria entra en la opción *Nuevo alumno* 2. La secretaria informa los datos del alumno 3. La secretaria guarda los datos del alumno haciendo clic en el botón guardar y volvemos a la lista de alumnos (ver caso de uso "Lista Alumnos")
Flujo alternativo	3. El sistema comprueba que todos los datos introducidos sean correctos. En caso contrario, informamos a la secretaria y le pedimos que corrija el error. 3. La secretaria *Cancela* el alta del alumno.
Postcondición	Se ha creado un nuevo alumno en el sistema

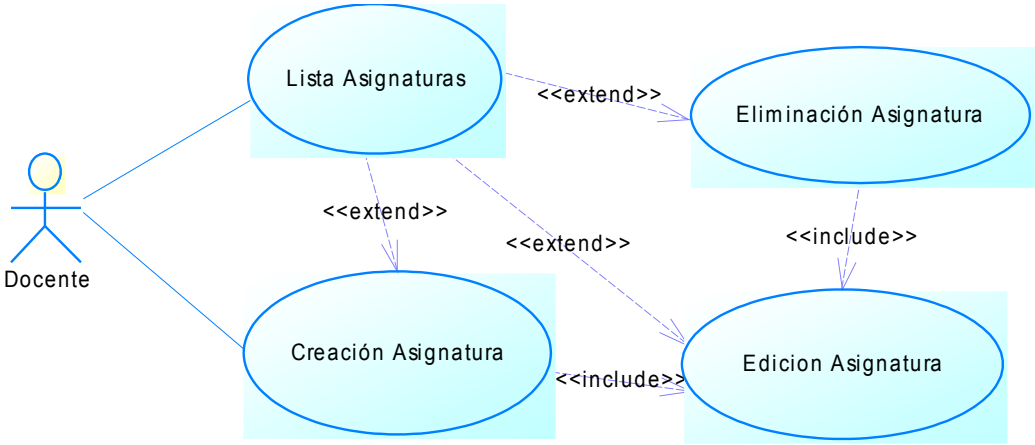
Caso de uso número 3: "Edición Alumno"	
Resumen de la funcionalidad	Modifica los datos de un alumno
Actores	Secretaria
Casos de uso relacionados	Lista Alumnos
Precondición	El alumno a modificar existe en el sistema
Flujo normal	1. La secretaria entra en la opción *Edición alumno,* previa selección del mismo (ver caso de uso Lista Alumnos) 2. El sistema muestra los datos relativos al alumno. 3. La secretaria edita los datos del alumno 4. La secretaria hace clic en el botón *Guardar* y los datos del alumno se guardan en el sistema
Flujo alternativo	4. La secretaria *Cancela* los posibles cambios, haciendo clic en el botón Cancelar
Postcondición	Se han guardado los datos del alumno en el sistema
Cuestiones a aclarar	El sistema en esta primera versión no contempla la eliminación de alumnos. En este caso de uso la secretaria podrá mantener un estado de alta/baja para indicar el estado del alumno. Sólo podrá dar de baja alumnos que no tengan una matrícula abierta. Es decir, si un alumno tiene una matrícula abierta, se deberá anular la matrícula y posteriormente dar de baja al alumno.
Caso de uso número 4: "Consulta Alumno"	
Resumen de la funcionalidad	Muestra los datos de un alumno
Actores	Secretaria
Casos de uso relacionados	Lista Alumnos
Precondición	Ninguna, es una consulta
Flujo normal	1. La secretaria entra en la opción *Consulta alumno,* previa selección del mismo (ver caso de uso Lista Alumnos) 2. El sistema muestra los datos relativos al alumno. 3. La secretaria hace clic en el botón *Salir* y volvemos a la Lista de alumnos
Postcondición	Ninguna, es una consulta

Módulo gestión de asignaturas

Este módulo permitirá la gestión de todas las asignaturas que ofrece el centro. Lo más relevante de una asignatura es el número de créditos que indica la

carga lectiva de la misma. Las asignaturas también guardarán un indicador de si esta ofertada o no.

Las asignaturas siempre estarán vinculadas a un curso, es decir, los alumnos se matricularán de cursos y el centro académico ofrecerá a los alumnos cursos.



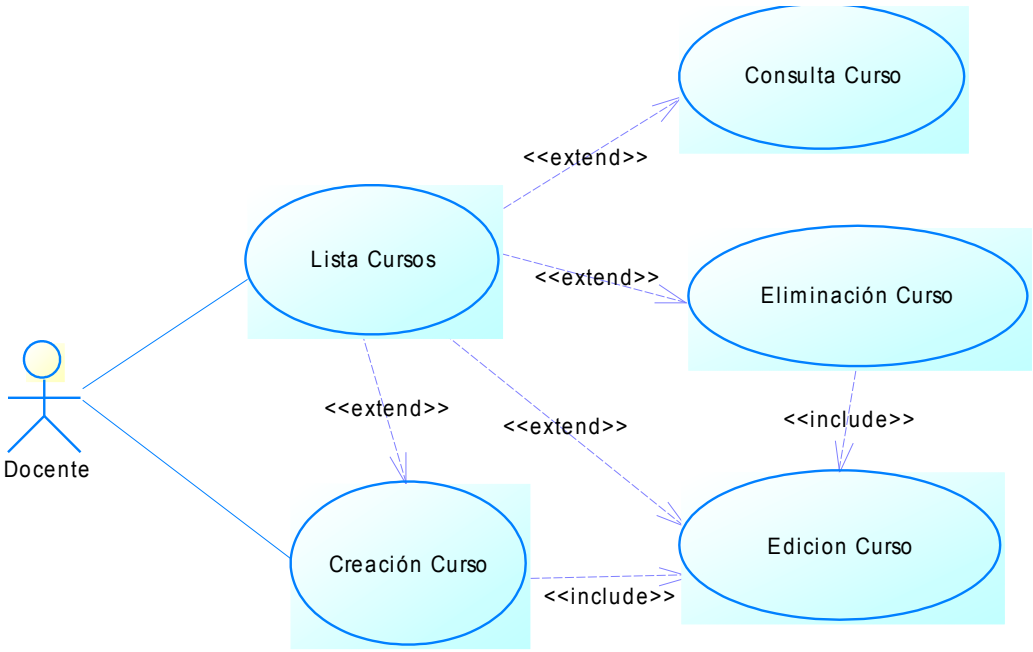
Caso de uso número 5: "Lista Asignaturas"	
Resumen de la funcionalidad	Muestra una lista ordenada por nombre, con los asignaturas que imparte o a impartido el centro
Actores	Docente
Caso de uso número 6: "Edición Asignatura"	
Resumen de la funcionalidad	Modifica los datos de una asignatura
Actores	Docente
Precondición	La asignatura existe
Cuestiones a aclarar	El personal docente pide clasificar las asignaturas en grupos de asignaturas. De todas formas, en esta primera iteración del proyecto no se contemplará esta opción y se dejará para una iteración posterior. Es decir, no se implementará en la versión final del TFC.
Caso de uso número 7: "Eliminación Asignatura"	
Resumen de la funcionalidad	Elimina una asignatura del sistema
Actores	Docente
Caso de uso número 8: "Consulta Asignatura"	
Resumen de la funcionalidad	Muestra todos los datos de una asignatura
Actores	Docente

Caso de uso número 9: "Creación Asignatura"	
Resumen de la funcionalidad	Da de alta una asignatura en el sistema
Actores	Docente
Cuestiones a aclarar	La misma que en el caso de uso "Edición Asignatura". ¿Es necesario agrupar las asignaturas en grupos? En caso afirmativo, tendremos que tenerlo en cuenta. De todos modos, de cara a la entrega final del proyecto está opción no estará implementada. La asignatura tendrá asociado un curso y solamente podrá pertenecer a un curso. En el futuro está previsto poder relacionar una asignatura a más de un curso. De todas formas, esta es una opción que el usuario considera secundaria en la primera versión del producto.

Módulo gestión de cursos

Este módulo permite la gestión de los diferentes cursos.

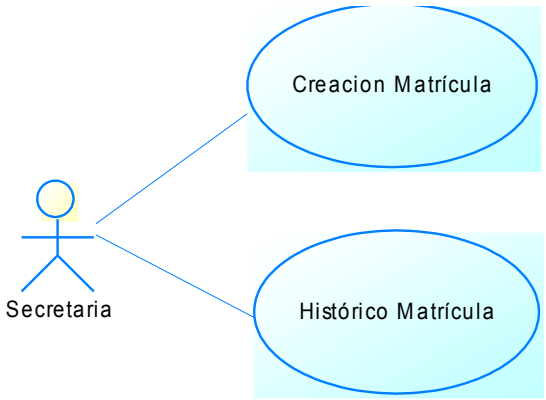
La parte más importante del módulo es la de relacionar las asignaturas a cursos. Una asignatura puede estar asociada a uno y solamente un curso. Un curso puede estar formado por varias asignaturas.



Caso de uso número 10: "Lista Cursos"	
Resumen de la funcionalidad	Muestra una lista ordenada por nombre de curso, con los cursos que imparte o a impartido el centro
Actores	Docente
Caso de uso número 11: "Edición Curso"	
Resumen de la funcionalidad	Modifica los datos de un curso
Actores	Docente
Cuestiones a aclarar	Un curso puede tener prerequisitos. Es decir, para cursar el curso Programación Avanzada, se debe haber realizado el curso Iniciación a la programación. De todas formas, esta opción se dejará para una iteración posterior y no se incluirá junto con la entrega final del proyecto.
Caso de uso número 12: "Eliminación Curso"	
Resumen de la funcionalidad	Elimina un curso del sistema
Actores	Docente
Caso de uso número 13: "Creación Curso"	
Resumen de la funcionalidad	Da de alta un nuevo curso en el sistema
Actores	Docente
Cuestiones a aclarar	La misma que en el caso de uso "Edición Curso" (ver caso de uso "Edición curso")

Módulo gestión de la matrícula

Este módulo permite formalizar la matrícula de un alumno a un curso determinado. El alumno siempre se matricula de un periodo determinado.



Caso de uso número 14: "Histórico Matrícula"	
Resumen de la funcionalidad	Muestra la lista de matrículas de un alumno
Papel dentro del trabajo del usuario	Caso de uso muy frecuente en periodo de matriculación
Actores	Secretaria
Precondición	El periodo de matriculación está abierto
Flujo normal	1. La secretaria selecciona un alumno de una lista ordenada por apellidos y nombre del alumno 2. El sistema muestra el histórico de matrículas de dicho alumno ordenada por fecha de matriculación en orden descendente 3. Si para el alumno seleccionado se puede crear una matrícula, el sistema lo informará
Cuestiones a aclarar	En principio consideramos que se podrá crear una matrícula si existe un periodo abierto para matriculación y el usuario ya no se ha matriculado. La funcionalidad de *anular* una matrícula no se contemplará en esta primera versión del proyecto.
Caso de uso número 15: "Creación Matrícula"	
Resumen de la funcionalidad	Creación de la matrícula de un alumno
Papel dentro del trabajo del usuario	Caso de uso muy frecuente en periodo de matriculación
Actores	Secretaria
Casos de uso relacionados	Histórico matrícula
Precondición	Se ha seleccionado un alumno en el caso de uso *Histórico Matrícula* El curso al cual se va a matricular el alumno está dado de alta en el sistema La matrícula a crear no existe El periodo de matriculación está abierto
Flujo normal	1. El sistema muestra el periodo donde se va a matricular el alumno. 2. La secretaria selecciona una lista con los cursos donde se puede matricular el alumno. 3. La secretaria selecciona la opción *Confirmar matrícula* y el sistema registra la matrícula. 4. El sistema genera el expediente académico del alumno. Básicamente es una lista de asignaturas del curso y la cualificación de dicha asignatura.
Flujo alternativo	3. La secretaria puede abandonar en todo momento el proceso

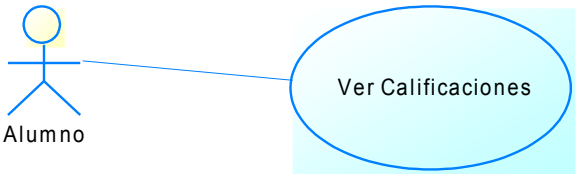
	mediante la opción *Cancelar proceso matriculación* 3. En caso de producirse algún tipo de error informaremos al usuario y mantendremos los datos que ha introducido hasta el momento.
Postcondición	El secretaria ha matriculado a un alumno de un curso
Cuestiones a aclarar	En esta primera versión del proyecto el sistema no comprobará que cursos son prerequisitos de otros. En un futuro podríamos restringir aún más la lista de cursos a matricular para que no aparecieran los cursos a los que no se puede matricular el alumno debido a que tienen otros que son prerequisitos de este.

Módulo gestión del expediente

Este módulo tiene una doble función.

Por un lado permite al profesor entrar las calificaciones de sus alumnos para una asignatura determinada. El profesor también tendrá la posibilidad de pedir una copia en formato portable (PDF) de las calificaciones que está editando.

Por otra parte permite a los alumnos ver sus calificaciones, así como pedir una revisión de las mismas sino está de acuerdo con alguna de las calificaciones. Los alumnos también podrán guardar una copia en formato PDF de sus calificaciones.



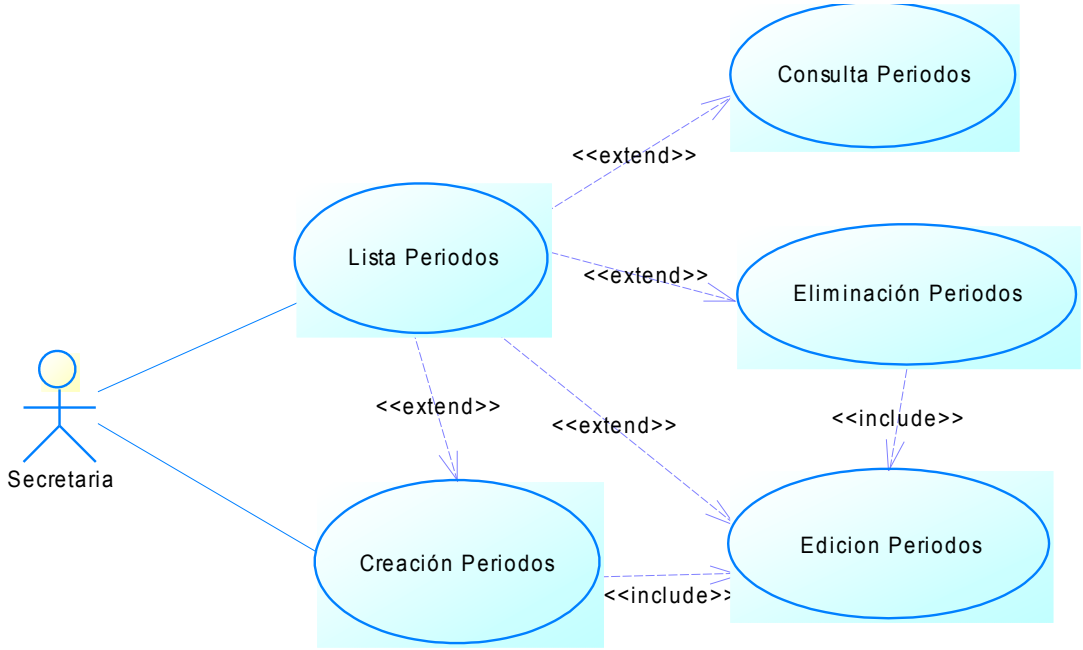
Caso de uso número 16: "Entrada Calificaciones"	
Resumen de la funcionalidad	Permite a un profesor entrar las calificaciones de un alumno en el sistema
Papel dentro del trabajo del usuario	Caso de uso muy frecuente cuando finaliza el curso
Actores	Profesor
Casos de uso relacionados	Revisar Calificaciones
Precondición	El periodo actual está abierto para la entrada de calificaciones
Flujo normal	1. El profesor entra en la opción *Entrada Calificaciones* 2. El profesor selecciona de una lista de asignaturas, una asignatura para la cual quiere ver las calificaciones 3. El profesor selecciona la opción *Entrar calificaciones* 4. Aparece una lista de todos los alumnos matriculados en la asignatura y sus calificaciones (si ya se habían entrado anteriormente) 5. El profesor entra las calificaciones y selecciona la opción *Guardar Calificaciones*.
Flujo alternativo	2. Si no hay un periodo abierto para calificaciones, al profesor le aparecerá un mensaje como "PERIODO CERRADO PARA ENTRADA DE CALIFICACIONES. CONSULTE CON ADMINISTRACION" (el propósito de este módulo es la entrada de calificaciones y no la visualización histórica por parte del profesor). 4. En cualquier momento el profesor puede seleccionar la opción *Salir*. 4a. El profesor puede pedir una copia en formato PDF de las calificaciones. 1. El profesor selecciona la opción *Generar Informe Calificaciones*. 2. El sistema genera un archivo PDF con las calificaciones de los alumnos en la asignatura seleccionada 5. Si se produce cualquier tipo de error al guardar las calificaciones, se informa al profesor y se le deben conservar los datos entrados hasta el momento.
Postcondición	Las calificaciones de los alumnos en la asignatura seleccionada se han modificado.
Cuestiones a aclarar	Cuando en un futuro se implemente la funcionalidad de poder asignar una asignatura a más de un curso, se deberá tener en cuenta en este caso de uso para poder seleccionar por Curso y Asignatura Los alumnos pueden pedir la revisión de su calificación sino están de acuerdo con la misma (ver caso de uso Ver Calificaciones). En

	una iteración posterior, permitiremos que el mismo profesor reciba un mail cuando un alumno haya pedido la revisión y desde la opción de *Entrada de calificaciones* podrá corregir (si es el caso) la calificación. Esta opción queda fuera del alcance de esta primera versión del proyecto. No se ha creído necesario que un profesor pueda consultar el histórico de calificaciones. De todas formas, se deja abierta la posibilidad de que en un futuro el profesor pueda ver el histórico de las mismas.
Caso de uso número 17: "Ver Calificaciones"	
Resumen de la funcionalidad	Permite a un alumno ver sus calificaciones (tanto históricas como las del período en curso) En el caso de las calificaciones del periodo en curso y en el que se haya abierto la opción de revisión de la calificación, el alumno podrá pedir la revisión de la calificación sobre la asignatura con la que no esté de acuerdo con la calificación
Papel dentro del trabajo del usuario	Caso de uso muy frecuente cuando finaliza el curso
Actores	Alumno
Casos de uso relacionados	
Precondición	Ninguna, es una consulta
Flujo normal	1. El alumno entra en la opción Ver Calificaciones 2. El alumno selecciona un período de la lista de periodos y selecciona la opción *Ver Calificaciones* 3. El sistema muestra al alumno una lista de asignaturas con su calificación 4. El alumno selecciona la opción *Salir*
Flujo alternativo	3a. El período elegido por el alumno es un periodo abierto para revisión y el alumno no está de acuerdo con alguno de sus calificaciones. 1. El alumno selecciona la opción *Pedir revisión* sobre la asignatura en cuestión 2. Continúa así con tantas asignaturas sobre las que quiera pedir revisión. 3b. El alumno selecciona la opción I*mprimir boletín Calificaciones*. 1. El sistema genera un archivo PDF con las calificaciones del alumno
Postcondición	El alumno puede haber pedido la revisión de alguna de las asignaturas si no estaba de acuerdo con la calificación y se trataba de un periodo abierto
Cuestiones a aclarar	De momento el alumno tendrá respuesta a su alegación (revisión) por un mail que le enviará la secretaria directamente. Una vez

	haya recibido el mail, si vuelve a consultar sus calificaciones, estas ya estarán actualizadas (si la alegación ha prosperado). El envió del mail no lo gestionará la aplicación, sino que será el propio sistema de mail que tenga el centro.

Módulo gestión de períodos

Permite la gestión de períodos académicos del sistema. La secretaria delimitará cada uno de los períodos mediante una fecha de inicio y finalización.



Caso de uso número 18: "Lista períodos"	
Resumen de la funcionalidad	Muestra una lista con los períodos del sistema
Actores	Secretaria
Casos de uso relacionados	
Precondición	Ninguna, es una consulta
Flujo normal	1. La secretaria entra en la opción *Gestión de Períodos* 2. Aparece una lista con todos los períodos ordenada por fecha de inicio del periodo 3. La secretaria puede desplazarse por la lista y hacer scroll para ver todos los períodos. 4. La secretaria hace clic en el botón salir.

Flujo alternativo	3. La secretaria hace clic en el botón *Consultar* para ver los datos detallados del período (ver caso de uso "Consulta Período") 3. La secretaria hace clic en el botón *Editar* para editar los datos del período seleccionado (ver caso de uso "Edición Período"). 3. La secretaria hace clic en "Nuevo Período" para *Crear* un nuevo período (ver caso de uso "Nuevo Período")
Postcondición	Ninguna, es una consulta
Caso de uso número 19: "Nuevo período"	
Resumen de la funcionalidad	Crea un nuevo período en el sistema
Actores	Secretaria
Casos de uso relacionados	Edición período
Precondición	El período a crear no existe en el sistema No hay un período "FUTURO" creado en el sistema. Por período "FUTURO", entendemos un período superior al actual. Es decir, sólo podrá haber creado un período más, al actual en el sistema. Por ejemplo, si el período actual es "OTOÑO 2008", podremos crear "PRIMAVERA 2009", pero no "OTOÑO 2009". Por período "ACTUAL", entendemos el período en el que la fecha actual está en el intervalo de su fecha inicio y fin.
Flujo normal	1. La secretaria entra en la opción *Nuevo período* 2. La secretaria informa los datos del período 3. La secretaria guarda los datos del período haciendo clic en el botón guardar y volvemos a la lista de períodos (ver caso de uso "Lista períodos")
Flujo alternativo	3. La secretaria informa una fecha inicio superior a la de finalización del período. 3a. El sistema informa del error a la secretaria 3. La secretaria informa una fecha de inicio de período que ya está comprendida en el intervalo de otro período (solapamiento de períodos). 3b. El sistema informa del error a la secretaria 3. La secretaria *Cancela* el alta del período
Postcondición	Se ha creado un nuevo período
Caso de uso número 20: "Edición período"	
Resumen de la funcionalidad	Edita los datos de un período
Actores	Secretaria
Precondición	El período a modificar existe en el sistema El período es actualizable. Sólo se podrá actualizar el período ACTUAL, y si existe, el período FUTURO (ver caso de uso "creación período" para más aclaración.

Flujo normal	1. La secretaria entra en la opción *Edición período*, previa selección del mismo (ver caso de uso "Lista períodos") 2. El sistema muestra los datos relativos al período. 3. La secretaria edita los datos del período 4. La secretaria hace clic en el botón *Guardar* y los datos del período se guardan en el sistema
Flujo alternativo	3. La secretaria informa una fecha inicio superior a la de finalización del período. 3a. El sistema informa del error a la secretaria 3. La secretaria informa una fecha de inicio de período que ya está comprendida en el intervalo de otro período (solapamiento de períodos). 3b. El sistema informa del error a la secretaria 3. La secretaria *Cancela* el alta del período
Postcondición	Se ha modificado el período en el sistema
Cuestiones a aclarar	Del período ACTUAL, no se podrán modificar las fechas del mismo, ni el indicador ABIERTO MATRICULACION (se asume que una vez iniciado, todos los alumnos ya están matriculados). Del período FUTURO, no se podrán modificar los indicadores de ABIERTO CALIFICACIONES, ni ABIERTO REVISIÓN (se entiende que tiene que ser el período ACTUAL para hacer esto).
Caso de uso número 21: "Consulta período"	
Resumen de la funcionalidad	Muestra los datos de un período
Actores	Secretaria
Precondición	Ninguna, es una consulta
Flujo normal	1. La secretaria entra en la opción *Consulta período*, previa selección del mismo (ver caso de uso Lista Períodos) 2. El sistema muestra los datos relativos al período. 3. La secretaria hace clic en el botón *Salir* y volvemos a la Lista de períodos
Postcondición	Ninguna, es una consulta
Caso de uso número 22: "Eliminación período"	
Resumen de la funcionalidad	Elimina un período del sistema
Actores	Secretaria
Precondición	El período a eliminar existe en el sistema No hay matriculas creadas para el período a eliminar
Flujo normal	1. La secretaria entra en la opción *Eliminar período*, previa selección del mismo (ver caso de uso "Lista períodos")

	2. El sistema muestra los datos relativos al período. 3. La secretaria selecciona la opción *Eliminar* y el período se elimina del sistema
Flujo alternativo	3. Al eliminar se produce un error en el sistema 3a. El sistema informa del error a la secretaria 3. La secretaria *Cancela* la eliminación del período
Postcondición	Se ha eliminado el período en el sistema

Escenario del proyecto

La aplicación se utilizará principalmente en un centro docente (escuela academia, etc.). El centro docente dispondrá de un PC tipo servidor capaz de soportar un servidor de aplicaciones como puede ser Jboss Application Server y un sistema gestor de base de datos relaciones (SGDBR) basado en MySQL Server.

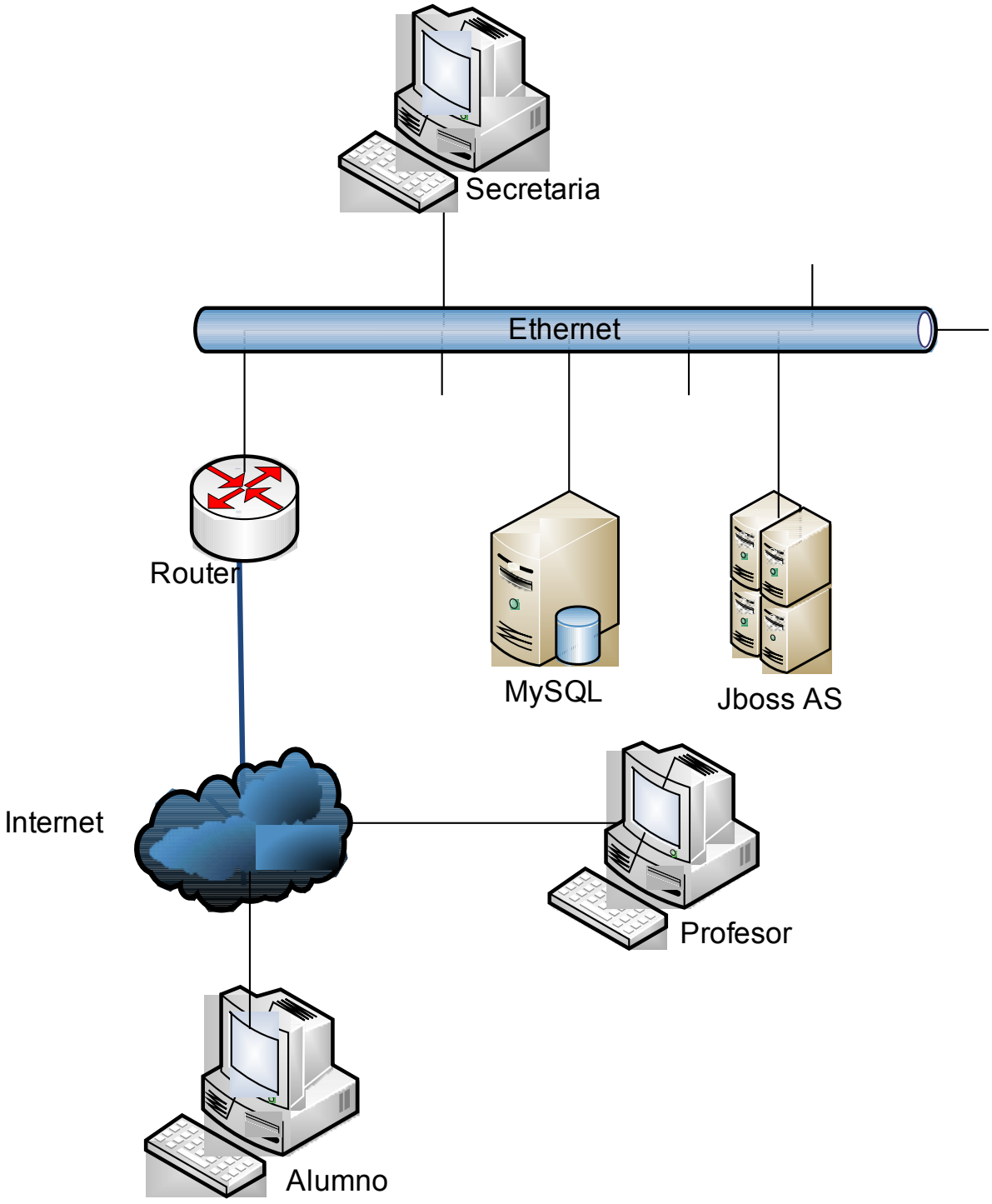
La aplicación se ejecutará en su totalidad bajo un entorno WEB, lo que facilitará el despliegue y las actualizaciones de las futuras versiones de la aplicación. El único requerimiento técnico en el cliente, es que tenga un navegador de última generación capaz de ejecutar llamadas asíncronas al servidor (lo que actualmente se conoce bajo las siglas AJAX).

Los actores que interactuarán con nuestro sistema son:

- *Secretaria*. Encargada de la gestión de alumnos, periodos lectivos, matriculación, etc..
- *Docente*. Encargado de la gestión de asignaturas y cursos.
- *Profesor*. Lo único que podrá hacer en el sistema es entrar las calificaciones de sus alumnos.
- *Alumno*. Puede consultar sus calificaciones así como pedir la revisión de su calificación sino esta de acuerdo con la misma.

Para cada uno de los actores, crearemos un Role en el sistema. Es decir, haremos una autentificación basada en Roles.

El esquema gráfico del escenario principal de trabajo es el siguiente:


Secretaria
Ethernet
Router
MySQL
Jboss AS
Internet
Profesor
Alumno

Diseño de la arquitectura de la aplicación

A continuación describiré como he estructurado en una arquitectura de capas la aplicación final. Distinguiré entre capas físicas (tiers) y capas lógicas (layers). Aunque esta diferencia no esta totalmente estandarizada, muchos autores[1] cuando se refieren a capas físicas utilizan el termino tier y cuando lo hacen a capa lógica lo hacen utilizando el termino layer. Para entendernos, una capa física normalmente se refiere a máquinas distintas y una lógica es una división en capas que normalmente se ejecutan en el misma máquina.

Capas físicas (tiers) de la aplicación

Como ya he mencionado, la aplicación será una aplicación totalmente en entorno WEB. Estas aplicaciones siguen un tipo de arquitectura Cliente/Servidor donde el cliente (navegador WEB) hace una petición al servidor, el servidor la recoge, realiza las acciones oportunas y devuelve el resultado (normalmente en forma de documento HTML) al cliente.

La comunicación entre la capa cliente y la capa servidor se realizará mediante Internet, con lo cual de esta infraestructura de comunicaciones no nos tenemos que preocupar porque ya está creada.

Por lo tanto, en nuestra aplicación tendremos dos capas físicas bien diferenciadas: capa cliente y capa servidor.

La capa cliente no nos importa mucho, porque el navegador WEB ya se encarga de facilitarnos la implementación de esta capa. A partir de ahora me centraré en como dividiremos de forma lógica la capa Servidor.

Capas lógicas (layers) de la aplicación

Por capas lógicas (layers) entendemos como dividiremos la capa del servidor en diferentes partes. En este proyecto, asumiré que cada una de las partes se ejecutará en el mismo servidor físico.

La división en capas lógicas tiene como objetivo principal el intentar desacoplar la funcionalidad propia de cada capa. Por ejemplo, si cambiamos nuestra capa de presentación de JSF a otra tecnología como puede ser Wicket, las otras capas no deberían verse afectadas o mínimamente afectadas por este cambio.

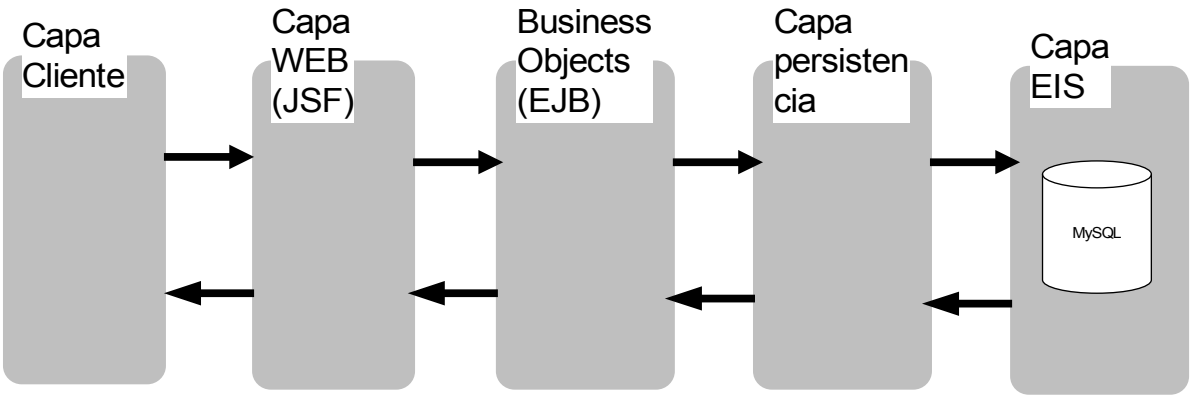
La división he capas lógicas que he efectuado es la siguiente:

- Servidor WEB
- Capa de componentes de negocio (business objects). Utilizaré EJB 3.0 para la implementación de esta capa.
- ORM (Object Relational Mapping). Capa de persistencia.
- Capa EIS (Enterprise Information System). Para esta capa tendremos un

1 Martin Fowler lo menciona en Patterns Of Enterprise Application Architecture

servidor de bases de datos relacional basado en Mysql.

El esquema siguiente resume de manera gráfica esta división en capas lógicas.



Para la *capa WEB* haremos servir el servidor HTTP que viene integrado con Jboss AS. Este servidor HTTP está basado en el servidor Apache HTTP Server. En la aplicación que se realizará, la capa WEB será la encargada de recoger las peticiones del usuario y realizar las llamadas oportunas a los componentes de negocio que se encuentran en la capa EJB, para finalmente devolver los resultados al cliente. La capa WEB no se comunicará directamente con la base de datos.

En la capa WEB, tendremos nuestra *capa de presentación* basada en *Java Server Faces* (JSF). Java Server Faces es un potente framework y que además tiene un gran soporte dentro de la comunidad de usuarios. Además, podemos utilizar multitud de componentes Open Source basados en JSF. De hecho, en este proyecto utilizaré Rich Faces como como librería de componentes JSF, que además soportan el uso de la tecnología AJAX.

Como *componentes de negocio* (business objects), utilizaré EJB 3.0. Realmente está es una decisión difícil de tomar porque como alternativa también tenemos a Spring Framework. Finalmente me he decantado por EJB 3.0 porque por lo que he podido investigar hasta el momento, la implementación es relativamente sencilla si lo comparamos con versiones anteriores. Además, por el hecho de usar EJB tendremos las siguientes ventajas.

- Son *transaccionales*. El contenedor CMT se encarga de su gestión.
- A*utentificación y autorización*. Igualmente el contenedor se encarga de la gestión de este aspecto.
- Mensajería gracias a los *Message Driven Beans (MDB)*. En este proyecto, no haré uso de los mismos, pero ofrecen gran potencia a las aplicaciones gracias al poder trabajar en diferido.
- *Job scheduler* gracias al uso de *timers*. Tampoco se van a usar en este

proyecto.

- Simplificación gracias al uso de *anotaciones*. Las anotaciones son un tipo de AOP (Aspect Oriented Programming), que aparecieron a partir de la versión 5.0 de Java.
- Es un *estándar* y por lo tanto, todos los servidores de aplicaciones certificados, los soportan.

He considerado el *ORM* otra capa más de la aplicación. La ventaja de utilizar un ORM reside en el hecho de que la transformación objeto/relacional y viceversa la hace el ORM. Nosotros trabajaremos con objetos JAVA en la aplicación. Las clases JAVA que formaran parte de nuestro ORM las diseñaremos a partir del modelo de dominio. El ORM que utilizaré en esa aplicación será JPA y como proveedor de persistencia Hibernate.

Como capa *EIS* utilizaré un potente gestor de bases de datos relacional (SGDBR) Open Source, como es MySQL. De hecho, podríamos utilizar cualquier tipo de SGDBR porque gracias al concepto de *dialecto* de Hibernate, podríamos cambiar de gestor de base de datos en cualquier momento si lo deseamos. Únicamente deberíamos cambiar el dialecto al de la nueva base de datos y nuestra aplicación continuaría funcionando correctamente.

Frameworks y patrones de la aplicación

Para el desarrollo de la aplicación Gestión Académica, utilizaré varios Frameworks. Los frameworks empleados tienen el propósito de simplificar el desarrollo de la aplicación.

Frameworks utilizados en la aplicación

SEAM Framework

La especificación JEE ya facilita de por sí el desarrollo de aplicaciones empresariales. Aún así, existen en el mercado Frameworks tanto Open Source como otros comerciales, que intentan agilizar aún más el desarrollo de aplicaciones y dotarle de más funcionalidad.

Este es el caso de SEAM Framework que utilizaré para el desarrollo del TFC. Al utilizar SEAM obtendremos las siguientes ventajas:

- Mejor integración entre JSF, JPA y EJB.
- Es un FRAMEWORK que utiliza un modelo de componentes contextual. En este sentido amplia el nivel clásico de scope (aplicación, sesión y request). En particular añade dos más: conversación y Business process.
- Nuestras clases de entidad sirven como back beans para JSF. Los back beans son unos clases Java que recogen los datos que son enviados por el cliente, generalmente mediante una operación POST de Http. Además, los back beans sirven como action listeners de las acciones que ejecuta el cliente.
- Uso de anotaciones para la configuración de componentes, en vez de XML

(aunque podemos hacer servir XML).

- Contexto de persistencia extendido. Esto nos permite mantener nuestros objetos mapeados por hibernate en la unidad de persistencia, incluso después de que finalice la fase de "request". Esto tiene la ventaja de evitar el famoso LazyInitializationException de JPA/Hibernate y de evitar la operación merge de JPA.
- Uso de AJAX a través de SEAM Remoting o con la integración con AjaxForJSF.
- Bien integrado con la librería avanzada de componentes RichFaces. RichFaces, son unos componentes basados en JSF.
- Generación de Reporting con PDF o EXCEL.

Hibernate

Hibernate es uno de los ORM y frameworks de persistencia más potente que existen en el mercado. A pesar de la curva de aprendizaje que supone llegar a conocer un framework de estas características, las ventajas de utilizarlo supera con creces esta curva de aprendizaje.

Además, Hibernate hace transparente el SGDBR que vayamos a utilizar, gracias al concepto de dialecto.

También ofrece características de optimización, como la cache de segundo nivel y otras muchas características, que sin la ayuda de este framework, con toda probabilidad las tendríamos que construir nosotros mismos.

Relacionado con Hibernate, también utilizaré el Framework Hibernate Validator. Este framework, nos permite anotar las entidades para dotarlas de validaciones básicas, como por ejemplo:

- @NotNull. El atributo no puede ser nulo.
- @Lenght (max=20). La longitud máxima del atributo debe ser 20.
- @Email. El atributo debe tener formato de Email.

Patrones y principios de diseño

Aunque Framework y patrón son conceptos distintos, tienen cierta relación. Normalmente los frameworks hacen uso de los patrones para su implementación. A continuación citaré algunos de los patrones de los que se hará uso en el proyecto como consecuencia del uso de los frameworks mencionados anteriormente.

- Active Record. Representa un registro de base de datos que tiene datos y comportamiento (ver http://martinfowler.com/eaaCatalog/activeRecord.html).

  SEAM Framework implementa este patrón en la clase EntityHome. Un

objeto tendrá la capacidad de guardarse, actualizarse, eliminarse, cargarse, validarse, etc..

- Factory Pattern. Este es un patrón creacional que resuelve el típico problema, "¿qué objetos tienen que crear otros objetos?". En SEAM gracias a la anotación Factory, será el propio Framework, quien, cuando le solicitemos un componente que no está creado, buscar un método que tenga esta anotación para construirlo.
- Inyección de dependencias. Es un tipo de implementación del principio de diseño "inversión del control". En este caso, tenemos un contenedor (SEAM) que mediante anotaciones nos inyecta los componentes que le solicitemos. Otros de los principios de diseño que sigue este patrón es el principio de Hollywood ("no nos llames, ya te llamaremos nosotros").
- *Carga parcial (lazy load).* Hibernate por defecto utiliza este patrón para las colecciones de elementos. Es decir, si tenemos una entidad que tiene una asociación con varios objetos de otra entidad, estos no se cargarán de base de datos, hasta que la aplicación los solicite.
- En la medida de lo posible, seguiré el principio de diseño *no te repitas a ti mismo*. Por ejemplo, al utilizar el framework *Hibernate validator*, estaremos siguiendo este principio.
- MVC. Este es el famoso patrón de diseño *Model View Controller*. En la aplicación el modelo serán nuestras entidades mapeadas con JPA y la vista serán las páginas Java Server Faces construidas utilizando facelets. En cuanto al controlador, o más bien, controlador de página, SEAM utiliza una tecnología llamada jPDL (Page Definition Language). Mediante este lenguaje podemos controlar el flujo de la página (estados y transiciones de la misma). Los controladores de página tienen el mismo nombre que la página JSF, pero con la extensión *Page*. Así, si nuestra página que muestra la lista de alumnos se llama alumnoList.xhtml, la página controladora se llamará alumnoList.page.xhtml. El controlador de página, además de gestionar el flujo entre páginas, también gestiona seguridad, autentificación, preparación del modelo, etc..

Diseño de la base de datos

A continuación se muestra el diseño de la base de datos.

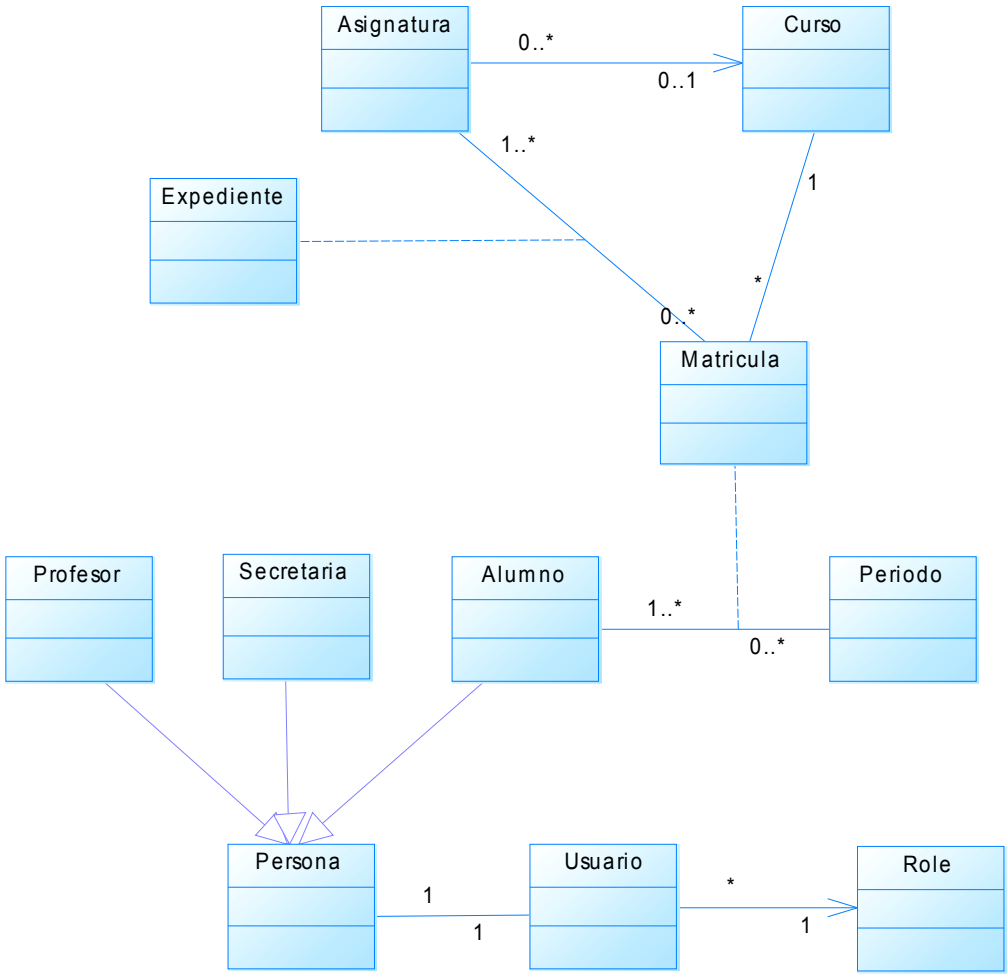
Primero se muestra el diseño conceptual. He elegido UML para el diseño del modelo conceptual, porque es más expresivo que el modelo relacional clásico.

Después se mostrará la transformación al modelo lógico relacional ya que es más cercano a la implementación en un sistema gestor de base de datos relacional.

Diseño conceptual

Como he mencionado antes, para el diseño conceptual utilizaré UML por ser un modelo mas rico que el modelo ER clásico.

El diagrama es el siguiente:



Diseño lógico

El diseño lógico lo haremos transformando el diseño conceptual confeccionado en el apartado anterior.

ENTIDADES

ASIGNATURA

(

id, nombre, numeroCreditos, ofertada, curso, version

ON {curso} REFERENCIA CURSO

)

CURSO

(

id, nombre, ofertado, version

)

PERIODO

(

id, nombre, fechaInicio, fechaFin, abiertoCalificaciones,

abiertoRevision, abiertoMatriculacion, version

)

ALUMNO

(

id, nif, fechaAlta, primerApellido, segundoApellido,

fechaNacimiento, direccion, poblacion, provincia,

estado, version

)

PROFESOR

(

id, nombre, primerApellido, segundoApellido, version

)

SECRETARIA

(

id, nombre, primerApellido, segundoApellido, version

)

INERRELACIONES

MATRICULA

(

id, alumno_Id, periodo_Id, curso_Id, estado, version

ON {alumno_Id} REFERENCIA ALUMNO

y {periodo_Id} REFERENCIA PERIODO

y {curso_Id} REFERENCIA CURSO

)

EXPEDIENTE

(

id, matricula_Id, calificacion, revisionPedida, estadoRevision,

version

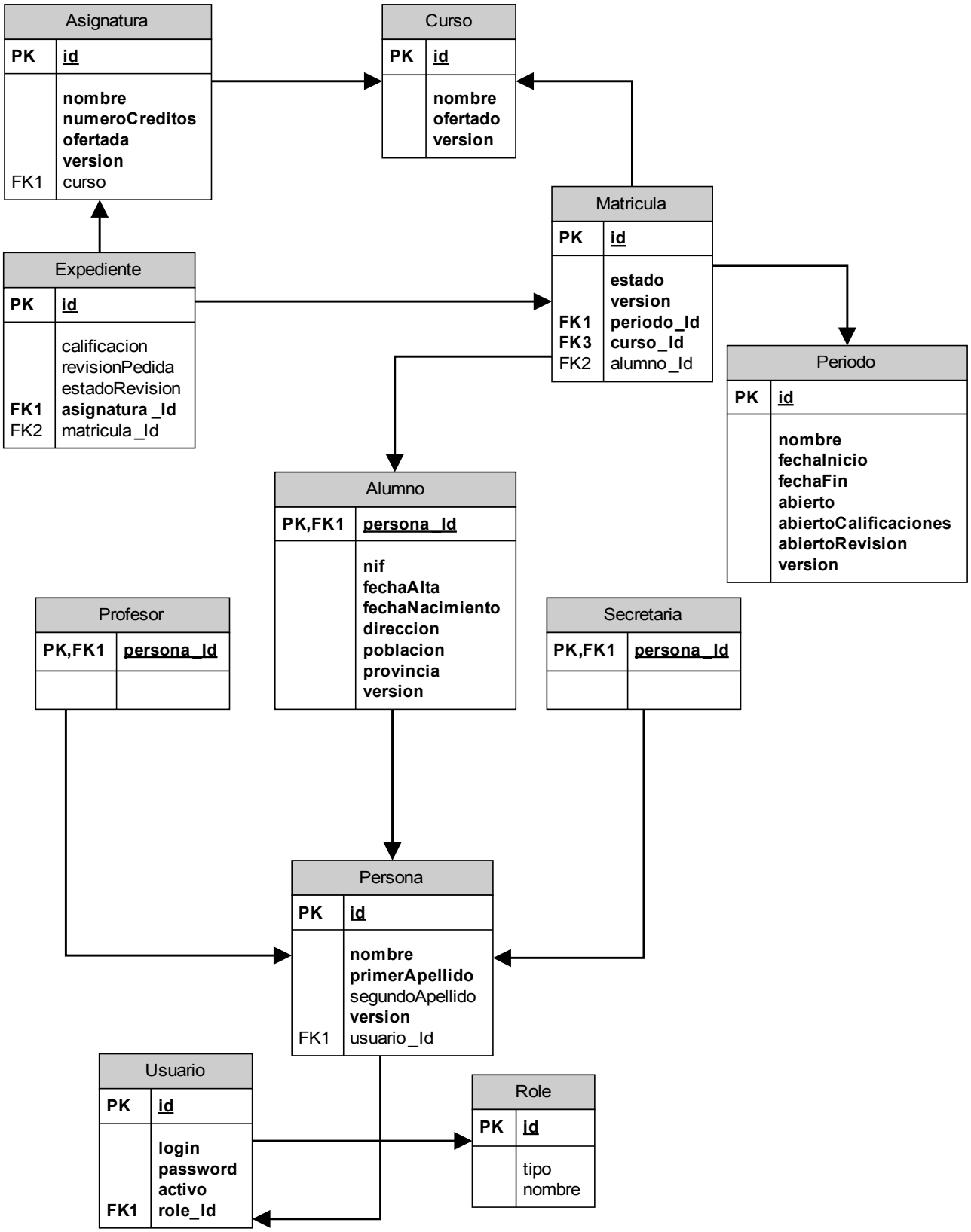
ON {matricula} REFERENCIA MATRICULA

)

Como se puede observar en el diseño lógico expuesto, para cada entidad he creado un identificador artificial. Esto es lo que comúnmente se denomina en base de datos como *surrogate primary key*. La experiencia nos dice que el utilizar un identificador artificial tiene múltiples ventajas:

- Simplifica la creación de las entidades.
- Simplifica el mantenimiento de las interrelaciones, y por lo tanto, actualizaciones en cascada.
- Ahorramos en espacio de almacenamiento, ya que si tenemos muchas interrelaciones, sólo tenemos que "arrastrar" el identificador artificial de la entidad.
- Simplifica el desarrollo de las aplicaciones que utilizan las entidades de la base de datos. Esto último es una consecuencia, puesto que el diseño de la base de datos no debería verse condicionado por la aplicación que la va a usar, ya que cada aplicación puede tener sus propias necesidades. Un ejemplo de esto es Hibernate. Aunque Hibernate soporta el uso de *primary keys* compuestas, el uso del mismo, dota de una mayor complejidad a la aplicación.

El diagrama correspondiente es el siguiente:


Asignatura
PK id
nombre
numeroCreditos
ofertada
version
FK1 curso
Curso
PK id
nombre
ofertado
version
Matricula
PK id
estado
version
FK1 periodo_Id
FK3 curso_Id
FK2 alumno_Id
Expediente
PK id
calificacion
revisionPedida
estadoRevision
FK1 asignatura_Id
FK2 matricula_Id
Periodo
PK id
nombre
fechaInicio
fechaFin
abierto
abiertoCalificaciones
abiertoRevision
version
Alumno
PK,FK1 persona_Id
nif
fechaAlta
fechaNacimiento
direccion
poblacion
provincia
version
Profesor
PK,FK1 persona_Id
Secretaria
PK,FK1 persona_Id
Persona
PK id
nombre
primerApellido
segundoApellido
version
FK1 usuario_Id
Usuario
PK id
login
password
activo
FK1 role_Id
Role
PK id
tipo
nombre

NOTAS

- PK indica clave primaria
- FK indica una clave externa o foreign key
- Los atributos en negrita son obligatorios
- Hay restricciones que el modelo no puede reflejar. Algunas las deberemos implementar en forma de triggers o procedimientos almacenados en la base de datos o como lógica de negocio de la aplicación. Por ejemplo, los periodos no se pueden solapar.

Diseño de clases y componentes

El proyecto lo he dividido en una serie de módulos, intentando que cada uno de ellos este mínimamente acoplado con los otros módulos. El tener un bajo acoplamiento es una gran ventaja en la programación en general y particularmente en la POO, ya que permite que podamos alterar un módulo sin verse afectados los demás (o al menos que el efecto colateral sea mínimo).

¿Por qué distinguir entre clases y componentes? Al utilizar el framework Jboss Seam, muchas clases que vamos a utilizar ya nos vendrán creadas. Únicamente las tendremos que parametrizar con nuestra funcionalidad propia. Una vez instanciadas ya tendremos disponible ese componente en un contexto determinado. Dicho de otro forma, un componente será la instancia de una clase determinada almacenada en un contexto de aplicación concreto (aplicación, sesión, conversación, página o evento).

Componentes Home

Como ya se ha descrito en el punto anterior, para la ayuda al desarrollo del TFC Gestión académica de un centro de formación, haré uso del framework JBoss SEAM. Una de las características que nos ofrece este framework son los componentes Home. Un componente Home viene a representar una entidad que se ha obtenido desde base de datos. Si nuestra entidad, representa una tabla de base de datos, entonces el Home será un registro de base de datos.

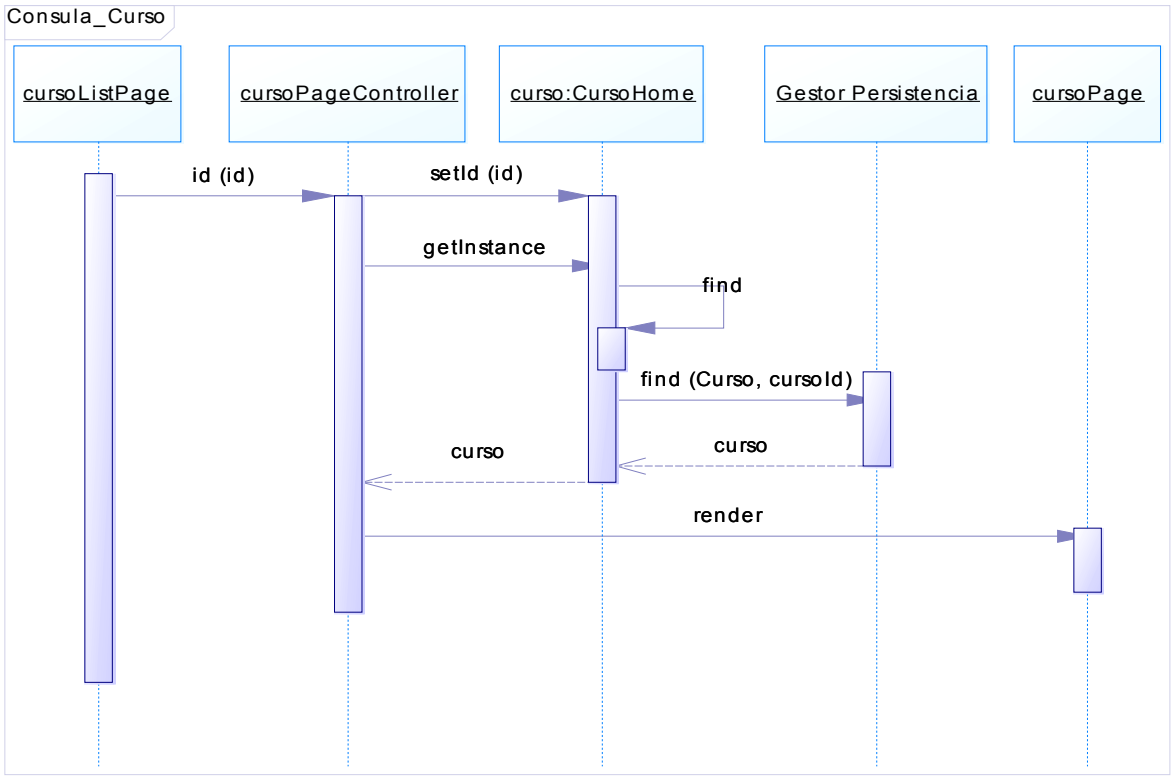
La clase Home se utiliza en SEAM para la creación rápida de CRUDs. En este proyecto utilizaré una clase especializada de Home, que se denomina EntityHome. Esta es una clase genérica, y le deberemos indicar con que entidad concreta vamos a trabajar.

Para el proyecto he extendido la clase EntityHome, dotándola de más funcionalidad. Esta clase la he llamado *TFCEntityHome*.

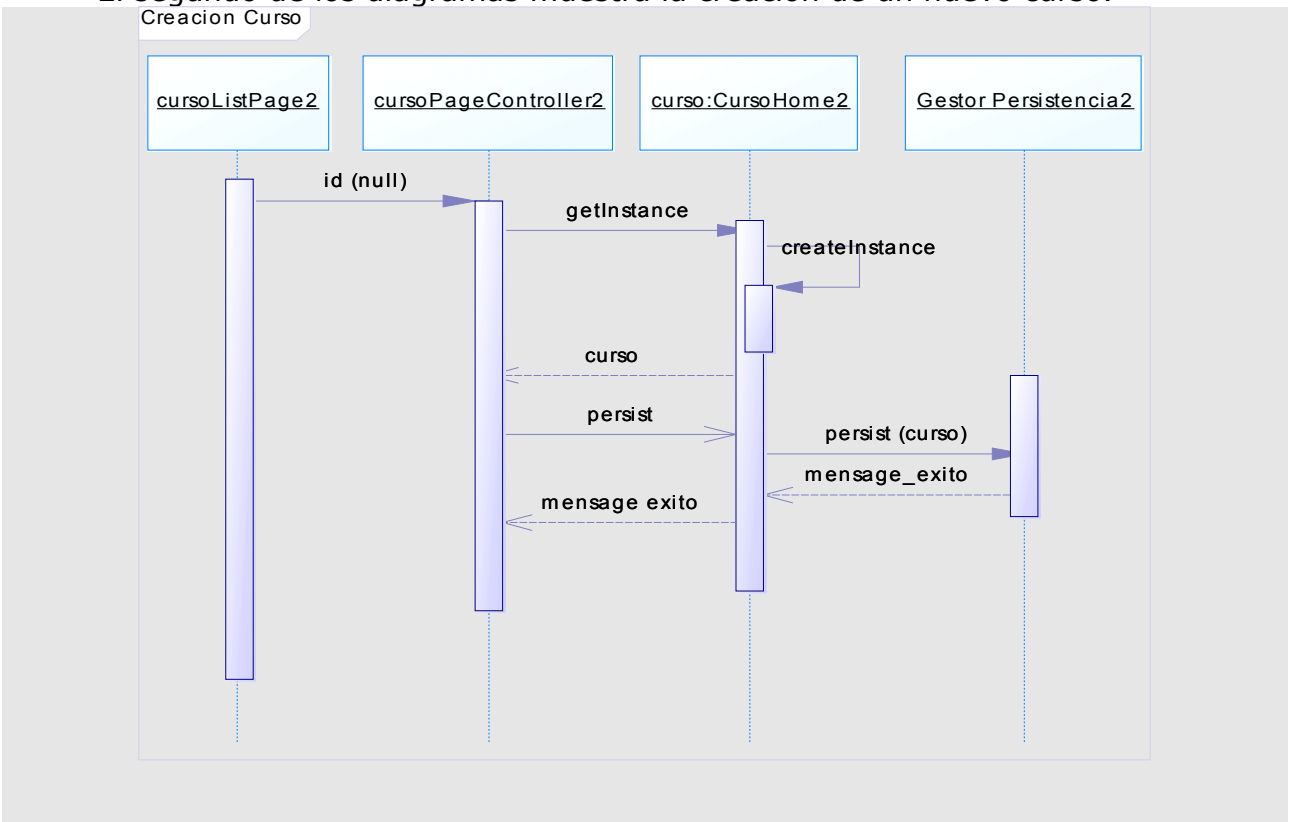
Explicación de los métodos existentes.

TFCEntityHome<E> extends EntityHome<E>	
Método	**Descripción**
+ update : String	Sincroniza la instancia con la base de datos. Es decir, si hemos modificado la instancia que contiene el EntityHome, la actualiza en base de datos.
+ persist : String	Almacena una instancia en estado *transient* en base de datos. Una instancia en estado transient en JPA es aquella que hemos creado y todavía no hemos almacenado en base de datos.
+ remove : String	Elimina el registro que representa el EntityHome de base de datos.
+ find :Asignatura	Recupera una instancia de la entidad de base de datos
+ isManaged : boolean	Indica si la entidad está en estado transient o en estado persistente. Por ejemplo, si hemos creado una nueva instancia porque vamos a dar de alta un nuevo registro en base de datos, este método nos devolverá false.
+setId (Object)	Le indicamos al EntityHome el id de la entidad que tiene que traernos de base de datos. Si vamos a crear una nueva entidad, no deberemos indicarle ningún id.
+ clearInstance	Hace un reset tanto del id como de la instancia del EntityHome.
+ getInstance : Asignatura	Devuelve la instancia que contiene el EntityHome. La primera vez que hacemos la llamada a este método, si hemos indicado un Id, utilizará el propio método *find* del EntityHome para cargar la instancia desde base de datos. Si no le hemos indicado Id, creará una nueva instancia de la entidad, con lo cual estará en estado transient. Es decir, el método isManaged nos devolverá false.
+setInstance (Asignatura)	Establecemos manualmente una instancia, sin utilizar el mecanismo de búsqueda por defecto del EntityHome.
+ revert	Anula cualquier cambio de la instancia y la solicita de nuevo la instancia al gestor de persistencia.
+ cancelPersists	Hace un clearInstance del EntityHome.

A continuación se muestran dos diagramas de secuencia donde el primero muestra un ejemplo de cómo usaríamos el componente EntityHome de la entidad curso para la consulta de los datos de un curso.



El segundo de los diagramas muestra la creación de un nuevo curso.



Componentes Query

Los componentes Query en SEAM permiten gestionar todo lo relacionado con las lista de objetos de una misma entidad.

Facilita la selección de objetos, paginación, ordenación, listas "clicables", cache de resultados, etc..

La clase específica para gestionar los objetos Query, que utilizaré en el proyecto será EntityQuery.

Este componente junto con el visto en el punto anterior (EntityHome), completan el desarrollo de CRUDs del TFC.

Al igual que con el objeto EntityHome, el objeto EntityQuery lo deberemos concretar para una clase específica.

Explicación de los métodos existentes.

EntityQuery<E>	
Método	**Descripción**
getEjbql : String	La consulta en formato Ejbql a ejecutar. Ejbql es un lenguaje muy parecido a SQL, pero para hacer consultas con nuestros objetos de entidad.
+ getRestrictions : List<String>	Aquí indicaremos los filtros a utilizar para seleccionar los datos.
+ getOrder : String	Orden en el que se mostrarán los resultados. Similar al *order by* de SQL.
+ getResultList : List<E>	La lista de objetos de nuestra entidad E, que cumple con los criterios indicados en las restricciones.
+ getResultCount : Long	El número de objetos que hay en el resultList.
+ refresh	Se vuelve a ejecutar la query contra la base de datos, para obtener los objetos actuales.
+getDataModelSelection : E	Dentro de los objetos que se hay en el objeto EntityQuery, SEAM es capaz de gestionar cual es el que el usuario ha seleccionado. Por ejemplo, cuando selecciona una acción en una tabla.
+getDataModelSelectionIndex : int	Igual que getDataModelselection, pero en vez de devolvernos el objeto seleccionado, nos devuelve el índice del resultList.

Si se observa el código fuente del producto final, no se verá ninguna clase Java que extienda de EntityQuery, o algún componente que instancie un componente EntityQuery. ¿Cómo entonces se crean estos componentes? En SEAM, tenemos un archivo de configuración llamado components.xml (ubicado en ProyectoWEB/

WeContent/Web-Inf). A continuación muestro un ejemplo de como crear un componente Query para mostrar la lista de asignaturas.

```
<framework:entity-query name="asignaturaQuery"
             scope="conversation"
             ejbql="select a from Asignatura a"
             order="a.nombre"
             max-results="5">
    <framework:restrictions>
        <value>lower(a.nombre) like lower( concat('%',
                    #{asignaturaExample.nombre},'%') )</value>
        <value>a.curso = #{asignaturaExample.curso}</value>
    </framework:restrictions>
</framework:entity-query>
```

No hay ningún inconveniente en utilizar código Java en lugar de XML. Tan sólo, he considerado que queda más accesible en el archivo XML.

Componentes Query By Example

Para parametrizar nuestro objeto EntityQuery utilizaremos el patrón *Query By Example*. Típicamente serán componentes instanciados a partir de nuestro modelo. Por ejemplo, para el caso anterior expuesto del componente *asignaturaQuery*, estamos utilizando el componente *asignaturaExample.* ¿Cómo se declara este componente? Lo podríamos declarar en el archivo *components.xml* o como finalmente he optado, en la propia entidad.

```
@Entity
@Name("asignaturaExample")
Roles({
      @Role(name="calificacionesAsignatura",
    scope=ScopeType.CONVERSATION)
})
public class Asignatura implements Serializable {}
```

Módulo gestión de alumnos

En este módulo nos encontraremos con las siguientes clases:

AlumnoHome extends TFCEntityHome<Alumno>

AlumnoQuery extends EntityQuery<Alumno>

Alumno
- id : Integer
- nif : String
- fechaAlta : String
- primerApellido : String
- segundoApellido : String

- fechaNacimiento : Date
- direccion : String
- poblacion : String
- provincia : String
- version : Integer

Módulo gestión de asignaturas

En este módulo nos encontraremos con las siguientes clases:

AsignaturaHome extends TFCEntityHome<Asignatura>

AsignaturaQuery extends EntityQuery<Asignatura>

Asignatura
- id : Integer
- nombre : String
- numeroCreditos : Integer
- ofertada : Boolean
- curso: Curso
- version : Integer

Módulo gestión de cursos

En este módulo nos encontraremos con las siguientes clases:

CursoHome extends TFCEntityHome<Curso>

CursoQuery extends EntityQuery<Curso>

Curso
- id : Integer
- nombre : String
- ofertado : Boolean
- version : Integer

Módulo gestión de matrícula

En este módulo nos encontraremos con las siguientes clases:

Matricula
- id : Matricula_Id
- curso : Curso
- estado : EstadoMatricula
- version : Integer

La entidad Matricula, tiene una clave compuesta formada por los atributos: curso, periodo y alumno.

Matricula_Id
- periodo : Periodo
- alumno: Alumno

GestorMatricula	
Interfaz local para la gestión de matrícula. Se provee una implementación en el programa, mediante el componente *GestorMatriculaBean*	
Método	**Descripción**
+ setAlumno (alumno)	Establece el alumno seleccionado por la secretaria, para el que se va a crear/consultar la matrícula
+ getAlumno : Alumno	Devuelve el alumno con el que está trabajando la secretaría
+ matricularAlumno : String	Matrícula a un alumno (el que está gestionando el componente) a un curso
+ creacionMatriculaDisponible : Boolean	Devuelve si para un alumno dado (el seleccionado) es posible crear una matrícula. Se podrá crear si hay un período de matriculación abierto y el alumno todavía no está matriculado de ese período
+getP*otencialMatriculacionCursosAlumno* *: List<Curso>*	Devuelve los cursos de los que se puede matricular el alumno

Módulo gestión del expediente

Expediente
- id : Expediente_Id
- calificacion: Float
- revisionPedida : Boolean
- estadoRevision : Integer
- version : Integer

La entidad Expediente, tiene una clave compuesta formada por los atributos: matricula y asignatura.

Expediente_Id
- matricula : Matricula
- asignatura : Asignatura

GestorExpediente	
Método	**Descripción**
+ getAsignaturas : List<Asignatura>	Devuelve una lista de asignaturas ordenadas por nombre. Utilizado en el módulo entrada de calificaciones para que el profesor pueda seleccionar una asignatura de una lista
+ getAsignatura : Alumno	Devuelve la asignatura seleccionada por el profesor para entrar las calificaciones
+ setAsignatura (Asignatura)	Establece la asignatura seleccionado por el profesor para entrar las calificaciones
+ expedientesAsignatura : List<Expediente>	Devuelve el expediente de todos los alumnos para una asignatura y el período actual
+ guardarCalificacionesAlumno : String	Guarda las calificaciones del alumno en el sistema. Si todo ha ido correctamente, redirige al profesor al informe de calificaciones
+guardarCalificacionesAlumno (Expediente)	Solicita la revisión de una asignatura, cuando el alumno no esta de acuerdo con ella
+ expedientePuedeRevisarse (Expediente) : Boolean	Comprueba si a la asignatura del expediente se le puede solicitar la revisión. Se podrá realizar si el período esta abierto, no se ha solicitado la revisión y hay una calificación de dicha asignatura

Módulo gestión de períodos

En este módulo nos encontraremos con las siguientes clases:

PeriodoHome extends TFCEntityHome<Periodo>

PeriodoQuery extends EntityQuery<Periodo>

Periodo
- id : Integer
- nombre: String
- descripcion : String
- fechaIncio : Date
- fechaFin : Integer
- abierto : Boolean
- abiertoCalificaciones : Boolean
- abiertoMatriculacion : Boolean
- version : Integer

GestorPeriodos	
Método	**Descripción**
+ periodoMatriculacionAbierto : Boolean	Determina si hay un periodo de matriculación abierto
+getPeriodoMatriculacionAbierto : Periodo	Devuelve el periodo abierto para matriculación si lo hay
+getPeriodoCalificacionesAbierto : Periodo	Devuelve el periodo abierto para entrada de calificaciones, si lo hay
+ getPeriodoActual : Periodo	Devuelve el periodo actual.Aquel cuyo intervalo de fechas está dentro de la fecha actual
+ getFechaInicioMinimaPeriodo : Date	Devuelve la fecha mínima a partir de la cual se puede crear un período para que no haya solapamiento de fechas
+isPeriodoEditable : Periodo	Determina si el periodo es editable (si esta abierto y su fecha inicio es posterior a la actual)
+ isPeriodoDeletable : Boolean	Determina si el periodo se puede eliminar (si su fecha inicio es posterior a la actual)
+periodoEsFuturo : Periodo	Determina si la fecha de inicio periodo es superior a la actual
+ crearPeriodoPermitido : Periodo	Determina es posible crear un período. Se podrá crear si su fecha de inicio es superior a la del día

Conceptos de arquitectura avanzados

Autentificación y autorización

A continuación describo la arquitectura seguida para la autentificación y autorización al sistema.

Para la autentificación y autorización he hecho uso de la simplificación que hace SEAM Framework sobre la arquitectura JAAS (Java Authentication And Authorization Service).

Autentificación

Por autentificación entendemos el conocer la identidad de la persona o sistema que intenta acceder a un recurso del sistema (página, método de un objeto Java, imagen, report, botón de una página, etc.).

Para la autentificación he hecho uso del recurso de Identidad JPAIdentityStore de SEAM. Este recurso consiste en tener almacenadas las credenciales del usuario en base de datos y acceder a ellas mediante JPA.

La autentificación a través de las páginas la conseguimos poniendo la siguiente condición en la definición de la página:

```
<?xml version="1.0" encoding="UTF-8"?>
<page xmlns="http://jboss.com/products/seam/pages"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://jboss.com/products/seam/pages
       http://jboss.com/products/seam/pages-2.1.xsd"
       login-required="true">
```

Cuando en el archivo de definición de página ponemos la condición *login-required="true"*, le decimos al framework que este recurso necesita autentificación. Es entonces cuando se invoca al componente encargado la autentificación al sistema, el componente *authenticator*. A continuación muestro el código fuente de este componente:

```
@Name("authenticator")
public class Authenticator
{
        @Logger Log log;
        @In Identity identity;// Componente SEAM que gestiona los roles de usuario
        // Componente SEAM que contiene las credenciales que ha entrado el usuario
        @In Credentials credentials;
        @In EntityManager entityManager;

        @Out(scope=ScopeType.SESSION)
        Usuario usuario;

    public boolean authenticate() {
        log.info("authenticating {0}", credentials.getUsername());

        Usuario usuario = autentificarUsuario();
        if (usuario.getRole().getTipo()==RoleType.ALUMNO)
               identity.addRole("alumno");
        else if (usuario.getRole().getTipo()==RoleType.PROFESOR)
               identity.addRole("profesor");
        else if (usuario.getRole().getTipo()==RoleType.DOCENTE)
               identity.addRole("docente");
        else if (usuario.getRole().getTipo()==RoleType.SECRETARIO)
               identity.addRole("secretario");
        else
               throw new AuthorizationException ("Role no permitido para esta aplicación");

        return true; // Autentificación realizada correctamente
    }

    /*
     * Obtiene el usuario de la base de datos si el usuario y password coinciden
     */
    private Usuario autentificarUsuario () {
        try {
            usuario = (Usuario) entityManager.createQuery(
                   "from Usuario where login = :login and password = :password")
                   .setParameter("login", credentials.getUsername())
                   .setParameter("password", credentials.getPassword())
                   .getSingleResult();
        }
        catch (NoResultException e) {
        throw new AuthorizationException ("Usuario o password incorrectos. Revíselos, por favor");
        }

         return usuario;
    }
}
```

Como se puede observar en el código fuente anterior, el método *authenticate*,

llama al método *autentificarUsuario*, que usa el componente *Credentials* de SEAM, para comprobar si las credenciales pasadas por el usuario son correctas. En caso contrario, el componente lanza una excepción del tipo *AuthorizationException*.

Si la autentifación ha sido correcta, guardamos el role del usuario en el componente *identity*.

¿Cómo sabe el framework que tiene que invocar a este método para realizar la autentificación? Si observamos el archivo de configuración de componentes (components.xml), encontraremos la siguiente configuración.

```
<security:identity authenticate-method="#{authenticator.authenticate}" remember-me="true"/>
```

Mediante esta línea le estamos diciendo al framework cual es nuestro método de autentifación.

Autorización

Por autorización entendemos si un componente del sistema tiene acceso a un recurso o no. La arquitectura seguida para el *TFC Gestión académica*, es la autorización basada en roles. La gestión de roles en SEAM, la gestiona el componente *Identity*. A modo de ejemplo, se muestra la construcción del menú principal de la aplicación.

```
<rich:toolBarGroup rendered="#{s:hasRole('docente')}">
       <s:button value="#{messages['gestion.cursos']}" view="/cursoList.seam" />
       <s:button value="#{messages['gestion.asignaturas']}" view="/asignaturaList.seam" />
</rich:toolBarGroup>

<rich:toolBarGroup rendered="#{s:hasRole('secreatario')}">
       <s:button value="#{messages['gestion.alumnos']}" view="/alumnoList.seam" />
       <s:button value="#{messages['gestion.periodos']}" view="/periodoList.seam" />
       <s:button value="#{messages['gestion.matriculas']}" view="/matriculaList.seam" />
</rich:toolBarGroup>

<rich:toolBarGroup rendered="#{s:hasRole('profesor')}">
       <s:button value="#{messages['gestion.calificaciones']}" view="/calificaciones.seam"/>
</rich:toolBarGroup>

<rich:toolBarGroup rendered="#{s:hasRole('alumno')}">
       <s:button value="#{messages['gestion.expedienteAlumno']}" view="/expedienteAlumno.seam" />
</rich:toolBarGroup>

<rich:toolBarGroup location="right">
    <s:button value="#{messages['gestion.home']}"  view="/home.xhtml" propagation="none" />
</rich:toolBarGroup>
```

Como se puede observar cada componente visual toolbarGroup, se renderiza si el usuario es de un Role determinado. Por ejemplo, en el caso del docente, podrá acceder a la gestión de cursos y asignaturas. Los demás botones, no serán visibles para su role.

También podemos restringir la ejecución de un método en concreto. Por ejemplo, en el componte gestorExpediente, tenemos el siguiente método anotado:

```
@Restrict("#{s:hasRole('profesor')}")
public void guardarCalificacionesAlumno () {
}
```

Transacciones

Al estar en un entorno EJB, el contenedor será el encargado de controlar todo lo relacionado con el entorno transaccional.

Sólo hay que realizar un mínimo de configuración dentro del archivo *components.xml*, para decirle a SEAM que el mecanismo transaccional lo gestiona el contenedor de EJB. Esto hay que hacerlo así, porque SEAM no está condicionado a la existencia de un entorno con contenedor de EJB. Podríamos utilizar SEAM en un Tomcat, por ejemplo.

De todas formas, si estamos en un entorno EJB, lo mejor es delegar la tarea transaccional al propio contenedor de EJB. Únicamente hemos de añadir está línea al archivo components.xml.

```
<transaction:ejb-transaction />
```

Cualquier excepción del tipo *system exception* (normalmente las que heredan de *RuntimeException*), abortará la transacción.

En el caso de una excepción de aplicación (normalmente las excepciones controladas de Java), podemos indicar en la propia clase de excepción que cuando se produzca este tipo de excepción, queremos hacer un *RollBack*. Por ejemplo.

```
@ApplicationException(rollback=true)
public class ApplicationException extends RuntimeException
```

Aislamiento

La estrategia de aislamiento que he decidido utilizar en la aplicación, es la conocida como *control de concurrencia optimista*. Este mecanismo consiste en comprobar al actualizar cualquier cambio en base de datos, si el registro cambio desde que se cargo desde base de datos. En caso afirmativo, el motor de persistencia (en este caso JPA) nos lanzará una excepción del tipo *javax.persistence.OptimisticLockException.*

Para la implementación de este mecanismo JPA se basa en el uso de columna especial que puede ser un timestamp o un autonumérico. Si esta columna tiene un valor diferente cuando vamos a guardar el registro, JPA nos informará mediante la excepción que he comentado antes.

En esta Entity, se ve a modo de ejemplo como anotando la columna con la anotación `@Version,` conseguimos un control de concurrencia optimista.

```
@Entity
@Name("cursoExample")
public class Curso implements Serializable {

	@Version
	private Date version;

	. . . . . .
}
```

La ventaja principal del control de concurrencia optimista sobre otros (como el pesimista) es que las aplicaciones son más eficientes y escalables. Es decir, la aplicación soportaría un mayor número de usuarios, sin verse afectado su rendimiento.

Java Persistence Query Language (JPQL)

En la aplicación *Gestión Académica*, no he utilizado ninguna consulta SQL. Todas las consultas han sido utilizando el mecanismo de JPA llamado JPQL. Entre las ventajas que ofrece este mecanismo, destaco las siguientes:

- Consultas parametrizadas.
- Evitamos ataques mediante la técnica SQL Injection.
- Las consultas son "más" orientadas a objeto.
- Uso de la cache del motor de persistencia (en este caso la cache de Hibernate). Esto permite mejorar el rendimiento del acceso a base de datos.

Además Hibernate incorpora el componente *Criteria* que extiende el concepto de JPQL. El componente *Criteria* estará disponible en la versión JPA 2.0. Mediante este componente podemos hacer consultas orientadas a objeto como las siguientes:

```
	public List<Expediente> expedientesAsignatura () {
		Periodo periodoActual = gestorPeriodos.getPeriodoActual ();

		expedientes=(List<Expediente>)getHibernateSession()
		.createCriteria(Expediente.class)
		.add (Restrictions.eq("asignatura", getAsignatura()))
			.createCriteria("matricula")
				.add( Restrictions.eq ("periodo", periodoActual) )
		.list();

	return expedientes;
}
```

Como se puede observar queda bastante claro que estamos consulta la entidad Expediente, filtrando por periodo y asignatura.

Tratamiento de excepciones

Con Java Server Faces, el tratamiento de excepciones es algo tedioso. Seam Framework simplifica un poco el tratamiento de las mismas mediante jPDL. En el archivo *pages.xml* se puede observar que cuando se produce una excepción, podemos redirigir el error a una página de error, que muestre un mensaje al usuario. A modo de ejemplo se muestra un fragmento del archivo *pages.xml*, donde hacemos el tratamiento de la excepción *javax.persistence.OptimisticLockException*, antes comentada:

```
<exception class="javax.persistence.OptimisticLockException">
        <end-conversation/>
        <redirect view-id="/error.xhtml">
        <message severity="warn">
                Otro usuario actualizado los mismos datos
        </message>
         </redirect>
</exception>
```

Como se puede observar, cuando se produce la excepción antes citada, se finaliza la conversación y redirigimos al usuario a una página de error con un mensaje determinado.

Contextos

Los componentes los almacenamos en contextos. La especificación JEE define tres tipos de contextos (aplicación, sesión y request).

SEAM amplia estos contextos con el contexto llamado conversación. Una conversación equivale más o menos a un caso de uso.

Todo el proyecto gira en torno al uso de conversaciones.

Por ejemplo, cuando entramos en la lista de alumnos se crea una conversación. Cuando editamos un alumno, se crea una conversación anidada de la primera. Cuando guardamos el alumno, se elimina la conversación anidada creada y finalmente, cuando salimos de la lista de alumnos se destruye la conversación y todos los componentes que se han registrado junto a ella.

¿Cómo asociamos un componente a la conversación en curso? Mediante la anotación *@Scope*. A modo de ejemplo, se muestra el siguiente fragmento de código:

```
@Name("gestorAsignaturas")
@Scope(ScopeType.CONVERSATION)
public class GestorAsignaturas {}
```

Este tema es muy amplio y por falta de espacio no se puede abordar con todo detalle.

Internacionalización

Para la internacionalización de la aplicación, seguiré el estándar de JEE. Es decir, en la capa WEB declararemos unos archivos de texto que en el contexto de JEE se denominan r*esource bundles*. Para diferenciar entre distintos idiomas, cada archivo tendrá el sufijo del idioma que contiene. Por ejemplo:

gesacad_messages_es.properties (español)
gesacad_messages_ca.properties (catalán)
gesacad_messages_fr.properties (francés)
gesacad_messages_en.properties (inglés)

Estos archivos tienen la siguiente estructura:

```
curso.nombre= Nombre Curso
```

Para recuperar los mensajes, utilizaré un componente de SEAM, llamado *messages*. Por ejemplo, si queremos recuperar la cadena anterior desde una página Java Server Faces, haríamos lo siguiente:

```
#{messages['curso.nombre']}
```

Finalmente hay que enlazar el componente messages con nuestro resource bundle. Esto lo hacemos en el archivo de componentes (*components.xml*).

```
<core:resource-loader>
	<core:bundle-names>
		<value>gesacad_messages</value>
		<value>messages</value>
	</core:bundle-names>
</core:resource-loader>
```

Tratamiento de la herencia

Java soporta la herencia entre clases. Sin embargo, cuando comencé a diseñar las entidades me encontré con el problema en que *Alumno*, *Profesor*, *Secretario* y *Docente* heredan de la entidad *Persona*.

En base de datos, lo solucioné con el mecanismo de *una tabla diferente por entidad*.

JPA nos permite implementar esta estrategia declarando en la clase base (en este caso *Persona),* la siguiente anotación:

```
@Entity
@Inheritance(strategy=InheritanceType.JOINED)
public class Persona implements Serializable {}
```

En las clases hijas, no hemos de hacer nada especial. Solamente heredar de la clase base, como haríamos con cualquier jerarquía de herencia.

Reporting

En la aplicación se han incluido dos informes en formato PDF (portable document format): calificaciones por asignatura para el profesor y calificaciones para el alumno.

La tecnología que hay detrás de estos informes se denomina *iText*. i*Text* son una serie de componentes desarrollados en Java que permiten la generación de documentos PDF. Jboss Seam ha incluido una capa por encima de iText que aún simplifica más el desarrollo de reporting. Prácticamente es como desarrollar una página Java Server Faces, con acceso total a los componentes gestionados por SEAM.

Logging

La gestión de logs es una parte fundamental en cualquier tipo de aplicación sobre todo a la hora de poder depurar o monitorizar posibles errores de una aplicación.

El típico *System.out.println()* de Java no es suficiente porque no permite, por ejemplo, filtrar mensajes, auditar la fecha y hora en que se produjo, etc..

Para el TFC Gestión Académica haré uso de un proyecto de Open Source Apache llamado *log4j*.

De hecho, el Framework SEAM también hace uso de l*og4j* y además tenemos de base un componente ya creado y configurado. Para usarlo, sólo hemos de solicitarlo al contenedor de la siguiente manera:

```
@Logger Log log;
. . . . . .
log.info("usuario conectado {0}", credentials.getUsername());
```

Mensajes asíncronos

Cuando estaba testeando el primer CRUD que realicé, pude observar que cuando daba de alta un registro y volvía a la lista de registros, no aparecía el registro que acababa de añadir.

Una manera de solucionarlo, podía ser que el propio botón añadir notificase a la lista que se actualizase. Esta no me parecía la mejor opción porque acoplaba funcionalidad propia de la vista, con la de un controlador.

La mejor opción que encontré fue utilizar una variante del patrón observer. Mediante jPDL podemos lanzar un evento (mensaje) que un observador (si lo hay registrado) se encargará de tratar.
Por ejemplo, en la eliminación de una asignatura tenemos el siguiente flujo:

```
<navigation from-action="#{asignaturaHome.remove}">
    <rule if-outcome="removed" >
           <end-conversation />
           <raise-event type="asignaturaDeleted" />
           <redirect view-id="/asignaturaList.xhtml"></redirect>
    </rule>
</navigation>
```

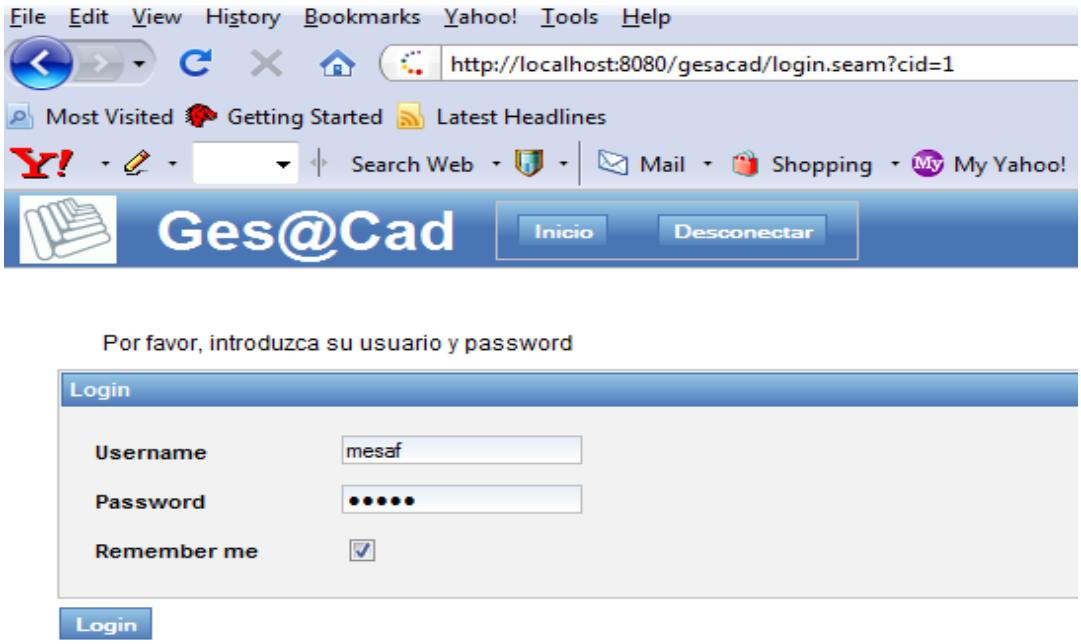
Como se puede observar cuando se elimina una asignatura, se lanza el evento *asignaturaDeleted*. ¿Quién trata este evento? El componente SEAM que tenga la anotación `@Observer("asignaturaDeleted")`. El método que trata este evento lo único que hace es "refrescar" la lista de asignaturas, de la siguiente forma:

```
@Observer("asignaturaDeleted")
public void asignaturaDeleted() {
       EntityQuery<Asignatura> asignaturaList =
              (EntityQuery<Asignatura>)Component.getInstance("asignaturaQuery");
       asignaturaList.first();
}
```

Pantallas de la aplicación

Para finalizar mostraré algunas de las pantallas de la aplicación.

Login



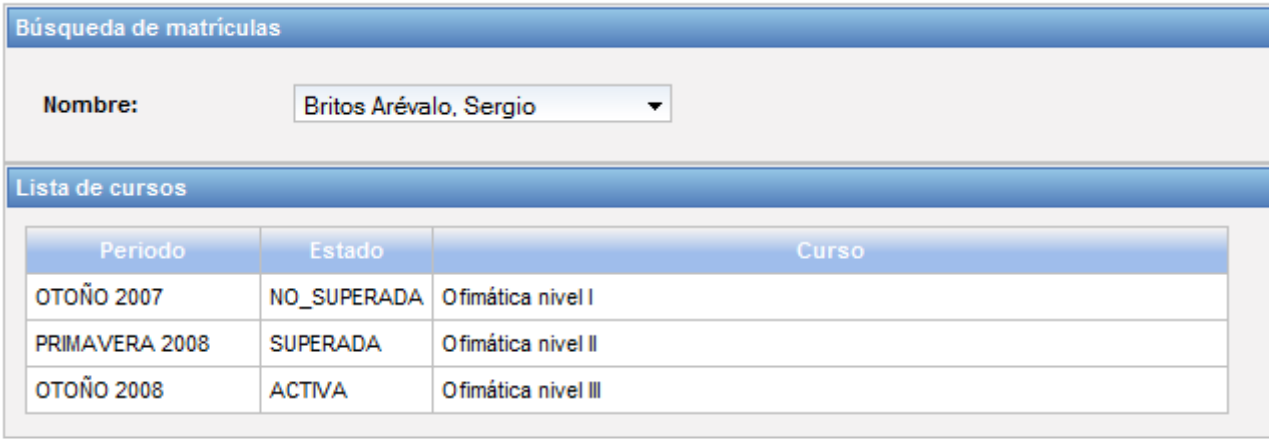
Lista de alumnos

Edición alumno

Lista de periodos

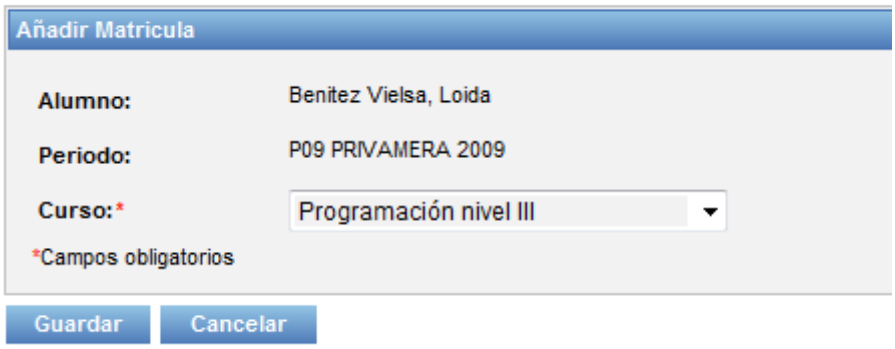
Fecha Inicio	Fecha Fin	Periodo	Descripción	A	AM	AC	AR	
04/09/2008	12/02/2009	T08	OTOÑO 2008	✓				
03/02/2008	03/07/2008	P08	PRIMAVERA 2008					
04/09/2007	12/02/2008	T07	OTOÑO 2007					
01/03/2007	05/07/2007	P07	PRIMAVERA 2007					
08/09/2006	13/02/2007	T06	OTOÑO 2006					

Histórico de matrículas

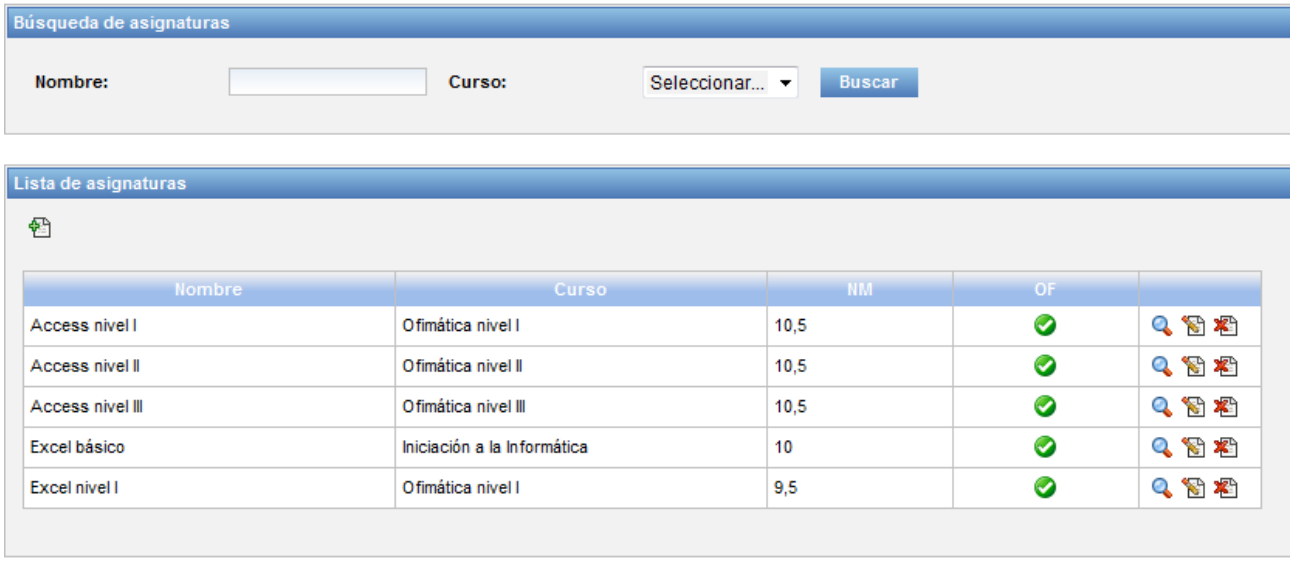


Periodo	Estado	Curso
OTOÑO 2007	NO_SUPERADA	Ofimática nivel I
PRIMAVERA 2008	SUPERADA	Ofimática nivel II
OTOÑO 2008	ACTIVA	Ofimática nivel III

Crear matrícula



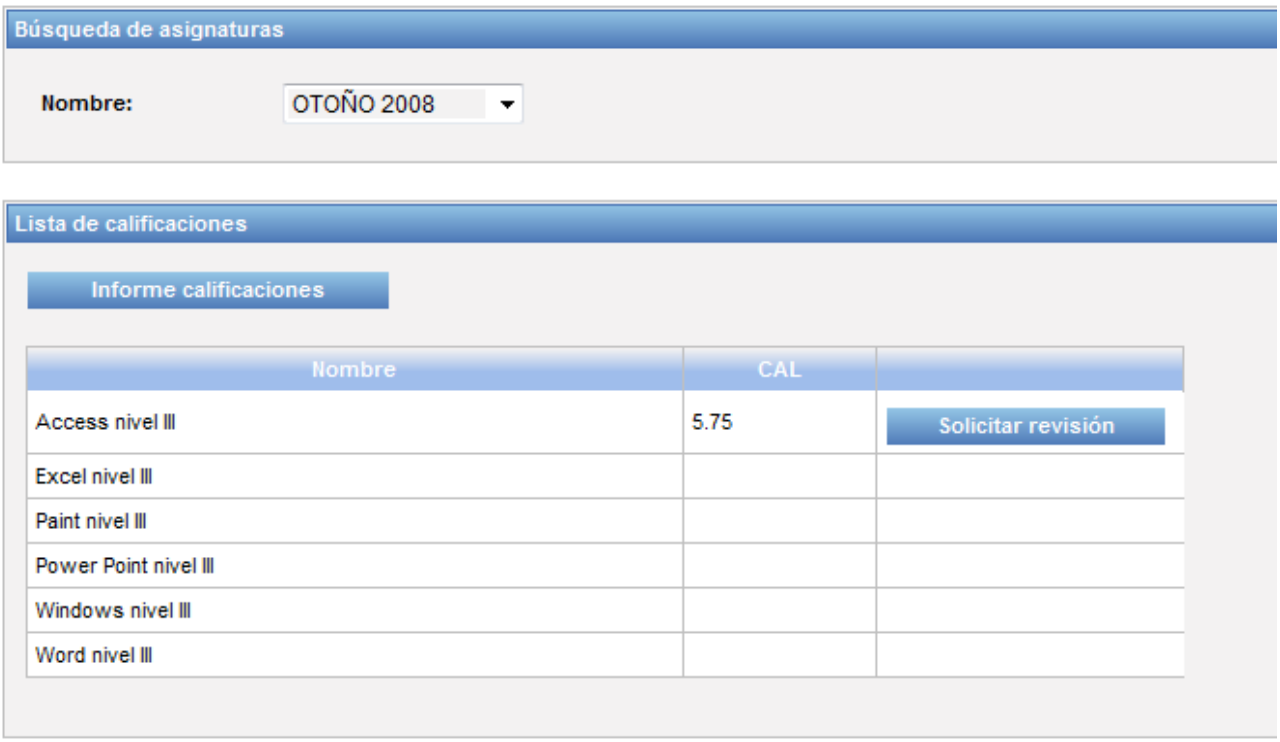
Lista asignaturas



Edición asignaturas

Entrada calificaciones (profesor)

Visualización calificaciones (alumno)



GLOSARIO

a4jsf
Framework que permite dotar a las aplicaciones JSF, de capacidades Ajax.

Alumno
Persona física que se ha dada de alta en el sistema como alumno y que puede matricularse de los cursos ofertados.

Asignatura
Material que se imparte en un curso.

Crédito
Unidad que indica la carga lectiva de una asignatura. A mayor número de créditos, mayor carga lectiva.

Curso
Un curso lo forman un grupo de asignaturas.

Docente
Persona encargada de la parte docente de la aplicación, principalmente asignaturas y cursos.

EJB
Enterprise Java Beans es la parte de la especificación JEE para poder crear la capa de componentes de negocio distribuidos y transaccionales de una aplicación empresarial.

Facelets
Mecanismo de renderización de páginas Java Server Faces y que se presenta como una alternativa a las tradicionales Java Server Pages (JSP).

Identity
Componente que forma parte del framework SEAM, encargado principalmente de guardar la identidad de un usuario (credenciales, roles, etc.).

iText
Tecnología basada en Java que permite crear documentos PDF de una manera sencilla.

JAAS
Acrónimo de Java Authentication And Authorization Service. Estándar JEE para la gestión de autentificación y autorizaciones.

JBoss
División de RedHat y encargada principalmente de la creación de MiddelWare. Es conocida principalmente por su popular servidor de aplicaciones llamado JBoss Application Server.

JBoss AS
Popular servidor de aplicaciones basado en la tecnología JEE y creado por Jboss.

JBoss SEAM
Producto creado por JBoss y que intenta ser un framework de integración de un conjunto de tecnologías basadas en Java principalmente, con el objetivo de simplificar su uso.

JEE
Acrónimo de Java Enterprise Edition. Estándar que establece una plataforma para desarrollar y ejecutar aplicaciones Java, con una arquitectura en capas distribuidas.

JSF
Java Server Faces es el la especificación recogida en el estándar JEE y que permite a los desarrolladores usar una serie de componentes ya creados y a los desarrolladores de componentes crear nuevos.

Matrícula
Asignación de un usuario a un curso, en un período determinado.

Periodo lectivo
División del tiempo en la que los alumnos se pueden matricular en un curso.

PDF
Acrónimo de Portable Document Format. Formato de documento muy extendido y propiedad de Adobe.

Profesor
Persona física que imparte asignaturas y que puede entrar en el sistema para asignar las calificaciones por asignatura.

RichFaces
Framework de componentes basado en JSF y actualmente propiedad de JBoss.

Secretario
Es la persona encarga de llevar a cabo las tareas administrativas del sistema (gestión de usuarios, matriculaciones, etc.).

BIBLIOGRAFIA

[1]IBM Developer Works (21/02/2006) Facelets fits JSF like a globe
http://www.ibm.com/developerworks/java/library/j-facelets/

[2]IBM Developer Works (19/08/2008) Developing Web applications with the Java Persistence API and JavaServer Faces
http://www.ibm.com/developerworks/rational/library/08/0819_mutdosch/

[3]Seam Framework (sitio oficial) Jboss SEAM Documentation
http://docs.jboss.com/seam/2.1.1.GA/reference/en-US/html/

[4]Sun Microsystems (08/2008) The Java EE 5 Tutorial
http://java.sun.com/javaee/5/docs/tutorial/doc/

[5] Dan Allen (2008). SEAM In Action. Manning

[6] Geary David, Horstmann Cay (2007). Core JavaServer Faces, 2nd edition. Prentice hall

[7] Geary David, Horstmann Cay (2007). Core JavaServer Faces, 2nd edition. Prentice hall

[8] Debu Panda, Reza Rahman, Derek Lane, (2007). EJB 3 In Action. Manning

[9] Craig Larman (2002). UML y Patrones, 2nd edition. Prentice Hall

[9] Bruce Eckel (2006). Thinking In Java, 4th edtion. Prentice Hall

[10] Christian Bauer, Gavin King (2006). Java Persistence with Hibernate. Manning

[11] Sun Microsystems. The Java tutorials
http://java.sun.com/docs/books/tutorial/

[12] Sun Microsystems. The Java tutorials
http://java.sun.com/docs/books/tutorial/

[13] Jboss RichFaces Documentación
http://www.jboss.org/file-access/default/members/jbossrichfaces/freezone/docs/devguide/en/html_single/index.html

ANEXO

Instalación de la aplicación

1) Crear base de datos y tablas en Mysql 5.1

1.1 Entrar en la consola de mysql (se encuentra en directiorio bin de la instalación de mysql) y ejecutar el siguiente mandato.

```
MySQL Server 5.1\bin>mysql –u root –p
```

1.2 Nos pedirá la password del usuario root e indicamos la password que tenga asignada el usuario root en esa instalación de MySql.

1.3 Crear la base de datos. Para ello ejecutamos el siguiente comando dentro de la consola de MySQL.

```
source c:/tfc/pac3/sql/database_creation.sql;
```

1.4 Poblar de datos la base datos, con algunos datos de muestra.

```
source c:/tfc/pac3/sql/database_populate.sql;
```

1.5 Si queremos eliminar los datos de la base de datos por algún motivo, ejecutaremos el siguiente mandato.

```
source c:/tfc/pac3/sql/database_drop.sql;
```

NOTA

La ruta indicada a Mysql, dependerá del directorio donde se hayan descomprimido los scripts.

Publicación en Jboss del proyecto

En el desarrollo de la práctica he utilizado el servidor ***Jboss As versión 4.2.3 GA.*** El EAR se ha generado teniendo en cuta este TARGET.

Publicación del DataSource

Junto con el zip que se ha adjuntado se entrega un archivo llamado *gesacad-ds.xml*. Este será el datasource a utilizar y preparado para acceder a MySQL. El contenido del mismo es el siguiente:

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

<!DOCTYPE datasources
    PUBLIC "-//JBoss//DTD JBOSS JCA Config 1.5//EN"
    "http://www.jboss.org/j2ee/dtd/jboss-ds_1_5.dtd">

<datasources>

   <local-tx-datasource>
      <jndi-name>gesacadDatasource</jndi-name>
      <connection-
url>jdbc:mysql://localhost:3306/gesacad</connection-url>
      <driver-class>com.mysql.jdbc.Driver</driver-class>
      <user-name>root</user-name>
      <password>passwordPersonal</password>
<!--
      <exception-sorter-class-name>
         org.jboss.resource.adapter.jdbc.vendor.MySQLExceptionSorte
rg
      </exception-sorter-class-name>
      <metadata>
         <type-mapping>mySQL</type-mapping>
      </metadata>
-->
   </local-tx-datasource>

</datasources>
```

La línea resaltada en color amarillo habrá que cambiarla con la password del usuario *root* de la máquina donde está instalado MySQL. También habrá que asegurarse que Mysql está a la escucha en el puerto 3306, sino también habrá que cambiarlo.

El DataSource lo ubicaremos en el siguiente directorio:

```
directorio_instalación_jboss\server\default\deploy
```

Instalación del driver JDBC en el servidor

Junto con el ZIP adjunto se facilita el jar JDBC para acceder a MySQL (mysql-connector-java-5.1.7-bin.jar).

Este JAR lo ubicaremos en el siguiente directorio:

```
directorio_instalación_jboss\server\default\lib
```

Publicación del proyecto de empresa EAR

Únicamente habrá que ubicar el archivo EAR que se ha adjuntado en el siguiente directorio:

```
directorio_instalación_jboss\server\default\deploy
```

El EAR se ha generado incluyendo los fuentes de Java.

Entrada en la aplicación

Una vez hemos publicado la aplicación, podremos acceder a la misma indicando la siguiente URL en el navegador web:

```
http://localhost:8080/gesacad/
```