REINGENIERÍA PRÁCTICA

DESARROLLO DE UN CAMPUS VIRTUAL INDEPENDIENTE DE LA PLATAFORMA A PARTIR DE UN CAMPUS EXISTENTE

Autores:
Daniel Martín Barrios
Javier Enrique Mendoza Gómez
Juan Antonio de la Vega Gutiérrez

TRABAJO FIN DE GRADO EN INGENIERÍA DEL SOFTWARE

Madrid, Junio de 2017

DIRECTOR:
Antonio Navarro Martín

Diccionario

- Business Object: Patrón que organiza el código de negocio en unidades lógicas que facilitan la mantenibilidad del sistema.
- CRUD: *Create, Read, Update and Delete*.
- DAO: *Data Access Object*.
- JAAS: *Java Authentication and Authorization Service* es una API de Java que se encarga de manejar los servicios de autenticación y acceso
- JPA: *Java Persistence API.* Es la API de persistencia de datos desarrollada para la plataforma JAVA EE.
- JSF: *Java Server Faces*. Es un marco desarrollado para aplicaciones Java basadas en tecnologías web.
- Outcome: Reglas de navegación usadas en la tecnología JSF
- SA: Servicio de aplicación.
- SRS: Software Requirements Specifications.
- XP: *Extreme Programing*. Es una metodología de desarrollo de la ingeniería del software.

ÍNDICE

Resumen

Los campus virtuales son usados como herramienta docente por alumnos y profesores. Estas herramientas se suelen utilizar para gestionar material docente. Sin embargo, cada plataforma tiene una interfaz muy diferenciada de las demás. Estas interfaces dependen por completo de un *Learning Management System* (LMS) complementado por un conjunto de aplicaciones que automatizan ciertos procesos de carga de datos no soportados por los LMSs (Huertas & Navarro, 2015).

El objetivo de este proyecto era rediseñar por completo la interfaz de un campus virtual capaz de abstraer las distintas plataformas que estén por debajo, y de esta manera tener un único campus virtual, con la misma interfaz. Además se ha llevado a cabo un proceso de reestructuración de este campus, no solo de la interfaz, sino también de toda la estructura de negocio. Este proceso facilitará futuras mejoras y extensiones. Tras todo este proceso el resultado es un campus virtual que es independiente de la plataforma que se despliegue por debajo, con una estructura multicapa.

Summary

Virtual campuses are used as a teaching tool by students and teachers. These tools are often used to share teaching material. However, each platform has a very different interface from the others. These interfaces depend on the entire Learning Management System (LMS), complemented by a set of applications that automate certain data loading processes not supported by LMSs (Huertas & Navarro, 2015).

The objective of this project was to completely redesign the interface of a virtual campus capable of abstracting the different platforms below, and thus having a single virtual campus, with the same interface. In addition a process of restructuring of this campus has been carried out. This process of restructuring adds and delete some code for presentation tier and business tier. This process will facilitate future improvements and extensions. After finish this project, the result is a virtual campus that is independent of the platform that is deployed below, with a multitier structure.

1. Introducción

1.1 Antecedentes

Los campus virtuales son aplicaciones pensadas para gestionar las actividades docentes entre profesores y alumnos (Huertas & Navarro, 2015). A pesar de ser herramientas extendidas en la mayor parte de instituciones docentes, por lo general, su arquitectura es bastante simple, ya que dependen por completo de un *Learning Management System* (LMS) complementado por un conjunto de aplicaciones que automatizan ciertos procesos de carga de datos no soportados por los LMSs (Huertas & Navarro, 2015).

Desde hace algunos años, algunos miembros del grupo de investigación en e-learning de la Universidad Complutense de Madrid, e-UCM, han centrado parte de su investigación en conseguir arquitecturas de campus virtuales que faciliten su mantenibilidad y extensibilidad (Navarro et al., 2012).

Como parte de este trabajo, Francisco Huertas desarrolló un conjunto de interfaces de programación capaces de aislar a los campus virtuales de los LMSs subyacentes. Estos interfaces, llamados, *Interfaces Canónicos* (Huertas & Navarro, 2015) fueron implementados para los principales LMSs como servicios web SOAP (Newcomer, 2002). También se construyó un campus virtual J2EE (Coward, 2014), denominado Campus Virtual AACV (*Arquitecturas Avanzadas de Campus Virtuales*, TIN2009-14317-C03-01), que hacía uso extensivo de los mismos.

Los objetivos del campus virtual AACV eran comprobar la viabilidad de uso de los interfaces canónicos, así como de sus implementaciones como servicios web. El campus AACV es plenamente funcional y ha estado en servicio durante varios cursos académicos. Sin embargo, se observó que era necesario llevar a cabo una mejora en la interfaz gráfica de usuario con el fin de mejorar la usabilidad del campus virtual AACV.

Con tal fin se propuso este trabajo que originalmente estaba concebido para cuatro alumnos y contaba con una fuerte componente dirigida por modelos. Por problemas de matrícula, al final sólo tres de los cuatro alumnos pudieron matricular el trabajo, por lo que se optó por rebajar los requisitos del mismo eliminando la componente dirigida por modelos.

Los alumnos conocían al profesor que proponía el trabajo ya que son alumnos del Grado en Ingeniería del Software, y habían cursado con el profesor las asignaturas de Ingeniería del Software en segundo, y de Modelado del Software en tercero.

1.2 Objetivos

Los objetivos de este trabajo tienen una doble naturaleza que podríamos denominar software y docente.

Desde el punto de vista de la naturaleza software del trabajo, el objetivo era proporcionar una nueva interfaz más usable al campus virtual AACV. La nueva interfaz debería construirse utilizando el marco oficial de Oracle para el desarrollo de interfaces gráficas de usuario J2EE llamado *JavaServer Faces* (JSF) (Geary & Horstmann, 2010) y debería utilizar el marco oficial Oracle para la autenticación y autorización de usuarios J2EE llamado *Java Authentication and Authorization Service* (JAAS) (Oracle, 2017).

Otro objetivo software secundario era la mejora de la arquitectura del campus AACV, definiendo mejor las responsabilidades de los servicios de aplicación del mismo (Alur et al., 2003).

Un objetivo claro del proyecto era reutilizar en su totalidad las implementaciones desarrolladas por Francisco Huertas para los interfaces canónicos. No obstante, debido al uso extensivo que desde la lógica de negocio se hacía de estos servicios web, era fundamental conocer el marco *Java API for XML Web Services (*JAX-WS) (Hansen, 2007).

Desde el punto de vista docente el trabajo tenía como objetivos que los alumnos fuesen capaces de utilizar los conocimientos adquiridos durante sus estudios en un contexto de equipo, y comprobar las capacidades para adquirir nuevos conocimientos de manera autónoma. Es evidente por tanto, que tanto desde los objetivos educativos, como desde los objetivos docentes, los alumnos tenían que aprender de manera autónoma JSF, JAAS y JAX-WS.

1.3 Plan de trabajo

El proyecto original, con una fuerte componente dirigida por modelos, partía de la base de utilizar un modelo de proceso iterativo incremental, como el *Proceso Unificado de Desarrollo* (Pressman & Maxim, 2014). No obstante, debido a que la componente dirigida por modelos se excluyó del proyecto, y dado que los alumnos tenían que aplicar tecnologías con las que no estaban familiarizados, lo que forzaba a tener reuniones semanales con el profesor, al final se optó por seguir una metodología ágil, como *eXtreme Programming*, XP (Pressman & Maxim. 2014). En este tipo de metodologías se opta por un modelo de entregas paulatinas en las que se muestran versiones funcionales de código que con el paso del tiempo se va enriqueciendo de funcionalidad.

Así, el plan de trabajo por meses fue:

- Septiembre y Octubre: Formación en JSF, JAAS y JAX-WS.
- Noviembre: Primeras versiones de la interfaz y funcionalidad básica.
- Diciembre y Enero: Estos meses estuvieron marcados por los festivos y la presencia de exámenes parciales.
- Febrero: Inclusión de JPA para persistir Usuarios, Cursos y Roles y las relaciones de

estos con la plataforma.

- Marzo: Mejora de la interfaz para adecuarla más a los marcos actuales y adición de los módulos que faltaban.
- Abril: Correcciones de interfaz y código para soportar todas las plataformas.
- Mayo: Realización de la memoria del trabajo.

Para el desarrollo de la aplicación se propuso una división de los miembros del equipo en especializaciones por capas, pero esto al final no se llevó a cabo por decisión unánime del grupo, ya que todos los miembros hemos pensado que era mejor repartir la carga entre todos de cada una de las capas para así aprender y poder ayudar a cualquiera de los miembros que necesite ayuda en un momento determinado del desarrollo.

2. Tecnologías y herramientas usadas durante el Proyecto

2.1 Introducción

La arquitectura de campus virtuales es un tema poco tratado en la literatura. Los trabajos más explícitos son precisamente los publicados por el director del trabajo (Navarro et al., 2012; Huertas & Navarro 2015).

Por tanto en vez de incluir un capítulo centrado en el estado del arte que sería básicamente lo contado en dichos artículos, hemos optado en esta memoria incluir un capítulo que proporcione una pequeña descripción de las principales tecnologías y herramientas utilizadas en el desarrollo de este trabajo fin de grado (TFG).

El capítulo incluye tanto tecnologías y herramientas con las que nos hemos familiarizado a lo largo de la carrera como aquellas que hemos tenido que utilizar por primera vez en este trabajo.

2.2 Tecnologías utilizadas que ya conocíamos antes del proyecto

Java

Java (Sun Microsystems, 1995) es un lenguaje de programación orientado a objetos diseñado por Sun Microsystems, que en la actualidad pertenece a Oracle Corporation.
Las características de este lenguaje son de sobra conocidas, por lo que no merece mayor explicación, más allá de la versión utilizada, que es la 1.8.

JPA

Java Persistence API, (JPA) (Sun Microsystems, 2006) es una marco de Java desarrollado para el mapeo objeto-relacional, es decir, simplificando mucho, para la persistencia de datos. El marco tiene su propio lenguaje de consulta para realizar algunas operaciones *Java Persistence Query Language* (JPQL) (Sun Microsystems, 2006). Esta API está definida en el paquete *javax.persistence*.

El marco permite el mapeo de clases y sus relaciones a tablas relacionales con claves extranjeras. También soporta herencia. De esta forma, a través de un conjunto de anotaciones, clases Java pueden ser consideradas *entidades JPA*, y ser mapeadas sus instancias a filas de tablas relacionales. En nuestro proyecto hemos decidido optar por la implementación *open source* de *EclipseLink* de JPA por varios motivos: (i) su fidelidad al estándar JPA; (ii) su facilidad de uso; y (iii) ya lo habíamos utilizado en el contexto de otras asignaturas, como Modelado de Software.

Bootstrap

Para el diseño de las páginas web hemos usado el marco *Bootstrap* (Twitter, 2011) basándonos en la normativa de diseño *Material Design* (Google Design) para adecuar la interfaz a los marcos actuales de presentación, con una interfaz sencilla e intuitiva.

JQuery

JQuery (JQuery Team, 2006) es un marco de JavaScript cuya funcionalidad permite la manipulación del modelo de las páginas, manejo de eventos y animaciones a nivel de página web. En nuestro proyecto no se usa JavaScript ya que se ha decidido intentar no usar este lenguaje.

No obstante, hemos usado JQuery para detalles sutiles de la presentación de la página, como por ejemplo, ocultar y mostrar elementos como la barra lateral.

2.3 Herramientas utilizadas que ya conocíamos antes del proyecto

Eclipse

Eclipse (Eclipse Foundation, 2004) es una plataforma software compuesta por una variada gama de herramientas de programación *open source*. Eclipse se podría definir como un entorno cuyo objetivo es el de desarrollar aplicaciones enriquecidas, dando la posibilidad de navegar entre clases java relacionadas. Esta plataforma ha desarrollado diversos entornos de desarrollo integrados (IDE en inglés), como el *Java Development Toolkit*.

También uno de los puntos a favor de este entorno es que cuenta con una extensa comunidad de usuarios, y esto nos facilitó mucho las cosas a la hora de desarrollar debido a

nuestra inexperiencia en tecnologías como JSF o JPA. Una de las páginas más conocidas es la comunidad de Stack Overflow (Stack Overflow, 2017), donde se pueden encontrar dudas y respuestas sobre Eclipse y otras herramientas y tecnologías.

Además Eclipse cuenta con la capacidad de instalación de una gama muy variada de *plug-ins* que amplían su funcionalidad. Véase como la opción de instalar un *plug-in* de *Subversive* (CollabNet, 2000) que nos permitió guardar nuestro proyecto en un repositorio sin necesidad de instalar otro entorno distinto.

A la hora del desarrollo de nuestra aplicación hemos usado *Eclipse Mars 2 (4.5.2)* como entorno concreto. La configuración de este entorno también ha sido algo dificultosa pero gracias a la ayuda del tutor hemos sabido solucionar los problemas de incompatibilidades que teníamos. Los miembros del equipo ya contábamos con experiencia previa en este entorno gracias a asignaturas como *Tecnología de la Programación*, *Ingeniería del Conocimiento*, *Ingeniería del Software* y *Modelado de Software*. Dentro de eclipse hemos tenido que configurar las propiedades del proyecto para añadirle como facetas JPA, JSF y otras configuraciones.

La versión de Java sobre la que se ha trabajado es la 8, sobre su *JDK 1.8*. Esto nos ha dado algún que otro problema con las bibliotecas que íbamos a usar para la capa presentación como JSF, que se solucionó gracias a unas instrucciones que nos facilitó el tutor en las que se detalla exactamente cómo arreglar ese error.

IBM RSA

IBM Rational Software Architect (IBM RSA) (Rational Software, 2006), es una herramienta CASE (*Computer Aided Software Engineering*) que permite el diseño, modelado y desarrollo de aplicaciones software. Ya utilizamos esta herramienta en la asignatura de *Ingeniería del Software* y hemos profundizado en ella en *Modelado del Software*. La herramienta necesita una instalación previa de Eclipse y de Java que permite el desarrollo de modelos representados mediante diagramas UML. Los modelos desarrollados se ajustan a la arquitectura multicapa (Alur et al., 2003; Fowler, 2002). IBM RSA permite tanto la generación de código a partir de diagramas, como la generación de diagramas a partir de código.

Hemos utilizado la herramienta para proporcionar los modelos de la aplicación. Si bien su uso iba a ser mayor, en contexto del marco de desarrollo dirigido por modelos (MDA) (OMG 2014), la baja de un compañero en el equipo inicialmente previsto, y dificultades añadidas debido a la complejidad del proyecto, forzaron la eliminación de la componente MDA de este trabajo.

Apache Subversion

Apache Subversion (CollabNet, 2000) es un software que se usa para la gestión del control de las versiones. Subversion permite a los usuarios guardar remotamente proyectos, facilitando la gestión sobre estos, es decir, dando la capacidad de editar y eliminar ficheros de un proyecto. También cuenta con una herramienta de gestión de conflictos en caso de haber en dos versiones coexistentes del proyecto un fichero con contenido distinto. En este proyecto hemos usado Subversion para la gestión de los diagramas que desarrollamos al principio para estudiar y entender el proyecto.
Los repositorios remotos de SVN nos fueron facilitados por la facultad.

MySQL

Structured Query Language, SQL (IBM, 1995) es un lenguaje que permite la creación y manipulación de datos persistentes estructurados acorde al modelo relacional (Edgar Frank Codd, 1970). Este lenguaje ha sido uno de los primeros lenguajes comerciales para el modelo relacional. Actualmente es uno de los lenguajes de base de datos más usados en el mundo.

Nuestro proyecto define un esquema que contiene los datos relativos a usuarios, contraseñas y cursos de forma similar al campus virtual de la Universidad Complutense de Madrid (Navarro et al., 2014), pero a diferencia de éste, el campus virtual desarrollado en este trabajo no tiene una aplicación de carga de datos de matrícula.

El sistema de gestión de bases de datos relacionales utilizado ha sido MySQL 5.2 (Oracle Corporation, 2010). Para conectar el código con la base de datos MySQL hemos usado un conector de *myql-connector-java-5.1.36* (Oracle Corporation, 2015).

Para trabajar con la base de datos hemos decidido optar por la herramienta *MySQL WorkBench 4.3* (Oracle Corporation, 2017), y así poder comprobar que está guardando los datos que deseamos, en nuestro caso, los usuarios, los cursos y las relaciones entre usuarios, cursos y plataformas. *MySQL WorkBench* es una herramienta visual, esta herramienta permite la administración de bases de datos, diseño de bases de datos, creación y mantenimiento para el sistema de bases de datos MySQL. Esta herramienta cuenta con una interfaz de usuario cómoda y sencilla para el usuario que permite visualizar los contenidos de las tablas y hacer consultas SQL de una manera rápida.

Google Drive

Google Drive (Google, 2012) es una herramienta web que permite almacenar archivos de una manera remota. Esta aplicación permite también un control de versiones y un control de

seguridad sobre las carpetas, de esta manera no todos los usuarios pueden acceder a todas las carpetas. Esta herramienta también cuenta con un editor de textos online, que a su vez dispone de un control de cambios y de un sistema de comentarios. Las cuentas para esta plataforma han sido las cuentas de correo que ha facilitado la facultad a cada uno de sus miembros (correo de la Universidad Complutense).

Este software nos ha resultado muy útil ya que era la herramienta mediante la cual íbamos guardando las versiones funcionales de nuestra aplicación. También se ha utilizado para la realización del contenido de esta memoria, ya que facilita en grandes medidas la comunicación y a su vez desarrollo de contenido entre miembros del equipo. Las distintas entregas que se han realizado de esta memoria se han entregado mediante Google Drive.

TeamViewer

TeamViewer (TeamViewer, 2005) es un software que permite el control remoto entre equipos. Su uso privado es gratuito. Está aplicación se ha usado en el proyecto como herramienta sustitutiva para el equipo donde originalmente se realizaban las demos, ya que debido a problemas hardware y software, este equipo dejó de funcionar. Este software se conectaba en el ordenador del tutor y remotamente se controlaba el equipo de uno de los miembros del equipo. De esta manera se podía comprobar el correcto funcionamiento de nuestra aplicación remotamente. Su uso es sencillo, se limita a introducir una clave y una contraseña del equipo que quieres controlar remotamente.

Skype

Skype (Microsoft, 2011) permite videollamadas a través de Internet. Este software pertenece actualmente a *Microsoft*. Skype ha tenido un gran peso en el desarrollo de la aplicación porque los miembros del equipo se comunicaban y se coordinaban a través de esta herramienta. También cuenta con la posibilidad de compartir pantalla, lo cual ha facilitado que todos los miembros del equipo estuvieran al día respecto a lo que se estaba modificando en ese preciso momento.

Para usar esta herramienta es necesario la obtención de una cuenta gratuita en su propia página web. Una vez hecho esto la aplicación es similar a cualquier aplicación de mensajería convencional como WhatsApp (Facebook Inc, 2010).

Microsoft Word 2016

Este software es un diseñador y modelador de texto, pertenece a *Microsoft*. Esta aplicación permite el desarrollo texto. En nuestro proyecto lo usamos para maquetar y dar formato

a esta memoria ya que es una herramienta fácil y sencilla de usar y a los problemas que presenta Google Drive con sus formatos.

2.4 Tecnologías utilizadas que no conocíamos antes del proyecto

Java Server Faces

Java Server Faces (JSF) (Sun Microsystems, 2004) es un marco desarrollado para aplicaciones Java basadas en tecnologías web. Su objetivo es simplificar el desarrollo de las interfaces de usuarios en las aplicaciones Java EE. Esta tecnología se basa en *JavaServer Pages* (JSP) (Sun Microsystems, 1999), lo que nos permite desplegar páginas web. Los objetivos de este marco son los de definir un conjunto de clases base de Java para gestionar los eventos de entrada y los componentes de una interfaz de usuario. Nos proporciona un modelo de *JavaBeans* para enviar eventos desde la interfaz del usuario a la parte del servidor. También permite definir APIs para la validación de entrada tanto en cliente como en servidor. Para la parte servidor-cliente define unos *outcomes* y un sistema de navegación de páginas en formato xml.

Un *Managed Bean* es un *bean* manejado por JSF. Un *bean* es un POJO (*Plain Old Java Object*), que contiene métodos *getters* y setters, y los métodos que unen la capa de presentación con la capa de negocio. La declaración de los *beans* se realiza en el fichero *faces-config.xml*. Para cada man*aged bean* debemos introducir el nombre del *bean*, la clase donde se encuentra la clase a considerar como *bean*, y el *scope* que define la instanciación de la creación del *bean*. En nuestro proyecto, todos los *beans* definidos tienen como *scope*, sesión ya que queremos que el *bean* viva hasta que el usuario abandone la aplicación.

En nuestro proyecto, JSF nos ha permitido definir la navegación entre páginas mediante *outcomes*. Esto facilita mucho la comprensión de que página está relacionada con otras páginas. Esta navegación está definida dentro del fichero *faces-config.xml*.
En dicho fichero, una regla de navegación está compuesta por:

- Un identificador de página (opcional). En él se define la ruta de la página de la que se van a capturar los eventos.
- Un *outcome*: Es una cadena. Si anteriormente no se ha definido un identificador de página, entonces esta cadena se compara en todas las páginas por si alguna la lanza.
- Una página de redirección, a la que se acude en caso de que el *outcome* definido anteriormente sea capturado.

También ha permitido la presentación de los datos de la aplicación en la parte web del campus. Esta tecnología la desconocíamos todos los miembros del grupo. La biblioteca utilizada ha sido *javax.faces-2.3.0-m06*, capaz de soportar la carga de ficheros desde el navegador.

JAAS

Java Authentication and Authorization Service (Oracle 2017) es una API de Java que se encarga de manejar los servicios de autenticación y acceso. La meta principal de esta API es separar los conceptos de autenticación del resto de la aplicación para hacerlos así independientes. También nos provee de un posible *login* en el que el usuario necesite ingresar sus credenciales. Dentro de esta API hemos usado *Login Modules* que son los encargados de gestionar los credenciales que nos han introducido, los validan para asignarnos un rol y en caso de no existir nos hace un *logout* de la plataforma.

En nuestro proyecto como capa y manejador de la autenticación para cuentas de usuario y roles hemos usado JAAS para tener el control sobre las páginas a las que puede acceder un usuario dependiendo de su rol en la plataforma. A través del documento *web.xml* de JAAS definimos la configuración del *login*, la página de bienvenida de la aplicación, los roles y los permisos de acceso de estos a las páginas. Definiendo en este documento la jerarquía de páginas a las que puede acceder un determinado usuario.

JAX-WS

Java API for XML Web Service (Sun Microsystems) es una tecnología para crear servicios web y clientes que se comunican a través de mensajes XML.

Los servicios web son una tecnología que permite la utilización de software de una manera remota e independiente de la plataforma que lo invoque. Esto ahorra el despliegue de la funcionalidad en máquinas remotas ahorrando así problemas de incompatibilidades o problemas de despliegue. Existen dos implementaciones principales de servicios web: los servicios web SOAP (*Simple Object Access Protocol*) y los servicios web REST (*Representational State Transfer*) (Hansen, 2007).

Los servicios web SOAP se caracterizan por el intercambio de datos en formato XML, este paradigma de mensajería carece de estado y puede ser utilizado para definir protocolos más complejos y completos. Estos servicios definen su estructura en un documento WSDL (Hansen, 2007) que básicamente describe la funcionalidad y los requisitos necesarios para poder invocarlos y las convenciones para representar la respuesta del servicio web.

La creación de un servicio web SOAP es complejo, pero toda esta complejidad es asumida por JAX-WS que nos permite crear servicios web tanto para clientes como proveedores de manera sencilla.

Los servicios web RESTful se caracterizan porque necesitan HTTP para poder intercambiar datos, eso sí, en cualquier formato (XML, JSON, etc.).

En nuestro proyecto usamos los servicios web SOAP ya que toda la información que enviamos y recibimos esta en formato XML
La decisión fue tomada en un principio ya que la manera de invocar a estos servicios es bastante más sencilla que la invocación a un servicio RESTful.

2.5 Herramientas utilizadas que no conocíamos antes del proyecto

Apache Tomcat

Apache Tomcat (ASF) es una implementación *open source* de las tecnologías *Java Servlet, JSP, Java Expression Language* y J*ava WebSocket* desarrollado por *Apache Software Fundation* (ASF). Apache Tomcat está desarrollado y mantenido por miembros de la ASF.

Catalina es el contenedor de servlets de Tomcat. Implementa las especificaciones de *Sun Microsystems* para servlet y JSP. Coyote es el conector de Tomcat que soporta el protocolo HTTP 1.1 como un servidor web. Esto permite que Catalina actúe como un servidor web. Jasper analiza los archivos JSP para compilarlos en código Java como servlets. En tiempo de ejecución Jasper detecta cambios en archivos JSP y los recompila.

En nuestro proyecto usamos Tomcat v8.0 y en la configuración *context.xml* definimos un elemento Realm que representa una base de dato" de usuarios, contraseñas y roles asignados a esos usuarios.

3. Discusión de los resultados obtenidos.

3.1 Análisis de la nueva interfaz de usuario

Para empezar el análisis de los resultados obtenidos en este trabajo vamos a ver las diferencias que hay con nuestra interfaz gráfica web con la que había implantada en el campus virtual previamente desarrollado. Se han llevado a cabo varias mejoras respecto a la interfaz que se nos presentaba. Gracias a asignaturas optativas como *Interfaces de Usuario* y tras el estudio que hemos realizado de estas, se ha decidido cambiar gran parte de la interfaz de usuario.

Empezaremos hablando del *login*, en la versión anterior este elemento web estaba situado en la esquina superior derecha de la pantalla, con un tamaño reducido. Figura 1.

Fig 1. Login del antiguo Campus Virtual.

En la Figura 1 podemos observar que el 90% de la página de inicio está en blanco, teniendo únicamente dos elementos web sobre los que interactuar. Hemos llegado a la conclusión de que en una página de *login* lo importante es que el formulario para introducir los credenciales tiene que resaltar, no tiene que tener elementos de distracción. Este formulario debería ser el centro de atención porque sin introducir las credenciales, no podemos empezar a usar nuestra aplicación. En la Figura 1 lo remarcamos en rojo. También vemos un desplegable en el que se nos permite elegir la plataforma sobre la que queremos hacer *login*, este desplegable ha sido cambiado, ya que el login en nuestra aplicación es independiente de la plataforma en la que el usuario esté registrado. En la base de datos de la aplicación se dispone de una tabla que almacena los usuarios, y de otra tabla para saber en qué plataforma está registrado el usuario. De esta manera hemos llegado a la conclusión de que la elección de la plataforma es algo posterior en nuestro flujo. Esto además, facilita en gran medida la experiencia de navegación del usuario. Aparte de por motivos de diseño tanto gráfico como arquitectónicos, hemos decidido cambiarlo porque en la interfaz antigua, si se introducen las credenciales y a continuación, antes de seleccionar el botón *Identificarse*, se selecciona la plataforma en la que se quiere trabajar, se pierden los datos del formulario.

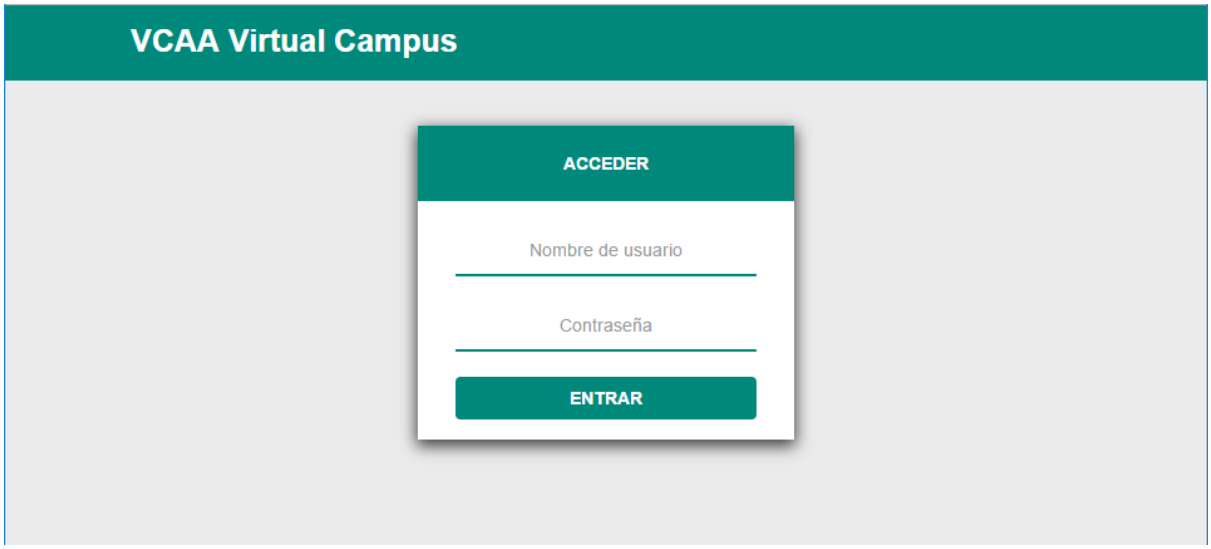

Fig 2. Login del nuevo campus.

En la Figura 2, podemos observar que la nueva interfaz se ha jugado con una gama de colores, en este caso verdes, blancos y grises. El formulario de *login* ha sido rediseñado para que sea el elemento importante dentro de esta página. El diseño del formulario se basa en el *Material Design* (Google 2015), ya que el dotar de sombra al formulario lo hace resaltar, hace que este elemento web parezca estar en un nivel superior comparado con el resto de elementos web.

Para el diseño del nuevo login de la aplicación se eliminó el campo de elección de plataforma. Ahora se ha rediseñado completamente el cambio de plataforma, después de haber pasado el *login* de una manera correcta, se mostrará una página intermedia entre el login y la página principal de la aplicación en la que el usuario podrá elegir plataforma.

En esta página los elementos importantes son los recuadros que contienen una imagen identificativa de la plataforma y un botón de acceso con el nombre de ésta. Figura 3. Se puede interactuar tanto con las imágenes como con los botones, estos botones son restrictivos. Que los botones sean restrictivos es por razones de arquitectura, ya que solo vamos a tener una plataforma desplegada por debajo, es decir, nos ha parecido más adecuada esta aproximación porque las plataformas no funcionan de la misma manera, a grandes rasgos son campus virtuales, pero a nivel de detalle, las tres plataformas usan validaciones de credenciales distintas, y cada plataforma te asigna un tipo de *token* que valida la sesión para poder hacer gestiones en la plataforma.

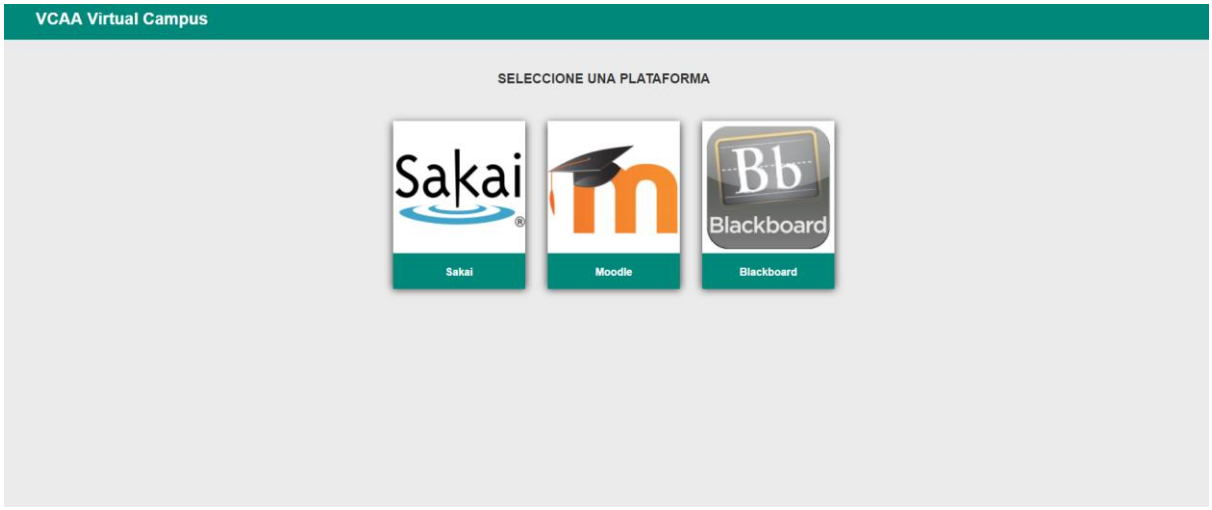

Fig 3. Selección de plataforma.

Aunque solo haya una plataforma desplegada en la aplicación, en cualquier momento de la navegación se puede cambiar de plataforma. Más adelante introduciremos este concepto cuando hablemos sobre el *header* que hemos definido para nuestra aplicación.

Además del rediseño del *login*, se ha decidido añadir un mensaje de error que aporte *feedback* cuando se han introducido unas credenciales que no son válidos. Figura 4. El diseño anterior no contaba con esta capacidad y si se introducían unas credenciales erróneas simplemente se recargaba la página sin aportar información de qué había pasado.

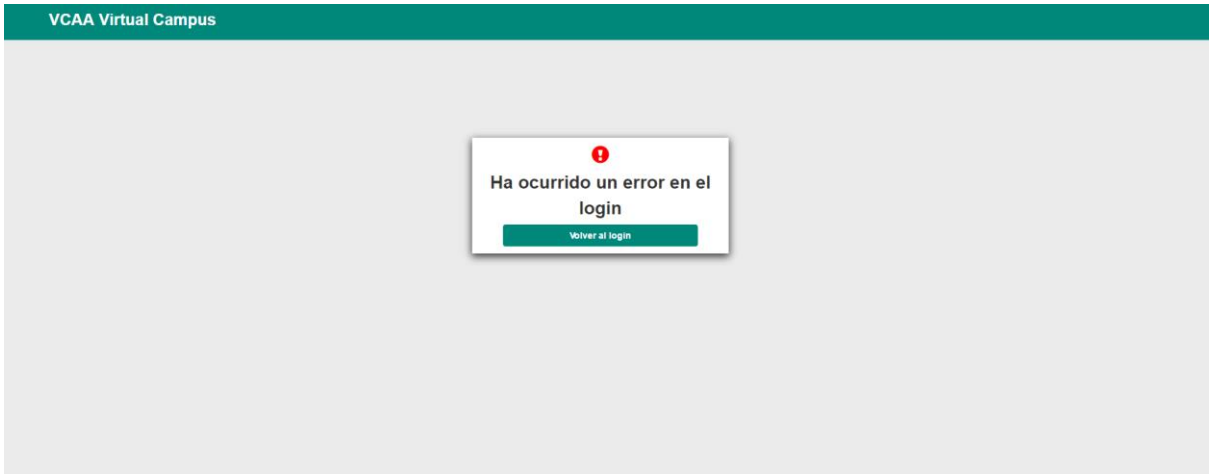

Fig 4. Error de login

Ahora se cuenta con un botón que redirige de nuevo a la página de *login* para permitir introducir de nuevo las credenciales. El mensaje de *feedback* está abierto a posibles mejoras, ya que por cuestiones de planificación no le hemos podido dedicar mucho tiempo, aunque creemos que está manera de aportar información acerca de si se ha hecho el *login* correctamente es importante. Esto también hace más agradable la experiencia de usuario que con la interfaz antigua

ya que si se fallaba a la hora de introducir las credenciales, podía quedarse esperando un tiempo indeseado.

Tras haber pasado en ambas interfaces de usuario el *login* se nos muestra una página de *home*.

Fig 5. Interfaz home del antiguo campus.

En la interfaz anterior se optó por una arquitectura dividida en pestañas. Figura 5. No hemos considerado que estuvieran bien identificados los elementos sobre los que podemos interactuar. Además al cambiar entre pestañas hay un pequeño tiempo de espera (desconocemos el porqué) pero en el nuevo diseño de campus no ocurre. En la interfaz previa, el elemento *header* cuenta con un desplegable que permite al usuario cambiar de plataforma, y con un segundo desplegable que le permite elegir el curso que desee gestionar. En la nueva interfaz, se ha decidido omitir este último desplegable, ya que no es necesario que haya varias maneras de poder acceder a un mismo curso puesto que esto puede provocar confusiones en la experiencia de usuario. Dado que se puede acceder a un curso a través de la pestaña de *Administración* (si se es admin) y desde *Perfil*, hemos creído que otra manera más de acceso a la misma página podía ser redundante.

En la nueva interfaz se ha optado por un header que permita volver a la página de *Home*, que permita volver a la página de *Administración*, y que también permita hacer un *logout* de la cuenta con la que hemos hecho *login*. Aparte de este desplegable, contamos con otro segundo desplegable que nos permite seleccionar en qué plataforma queremos desplegar por debajo.

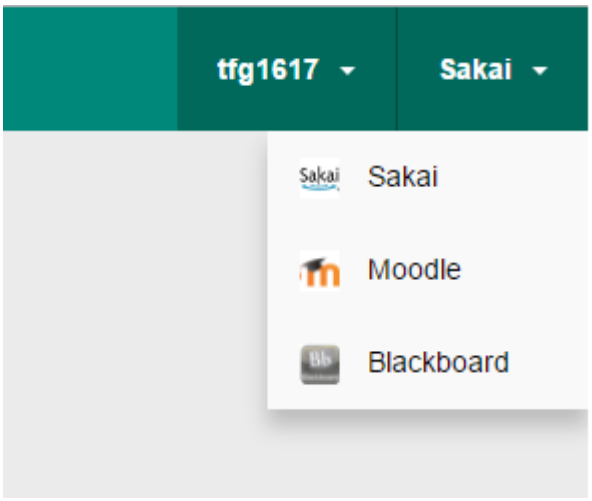

Fig 6. Desplegable cambio de plataforma.

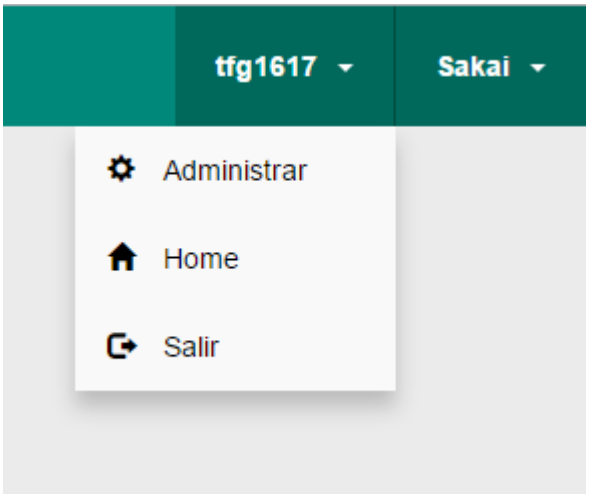

Fig 7. Desplegable de acceso rápido.

Estos desplegables se mostrarán al pasar el ratón por encima del nombre de usuario o de la plataforma activa. Figuras 6 y 7.

Toda la funcionalidad acerca de la navegación de nuestra aplicación se muestra a través de un sidebar (menú lateral). Figura 8. Por defecto el sidebar aparece desplegado pero se puede ocultar.

El contenido del sidebar cambia dependiendo del punto de la aplicación en la que se encuentre el usuario.

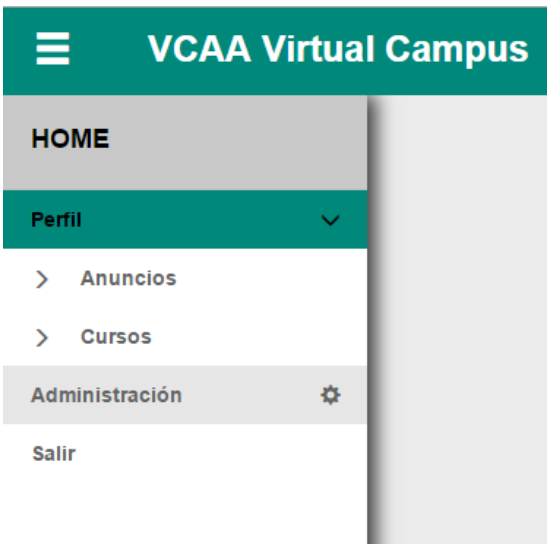

Fig 8. Sidebar de administración.

A diferencia del anterior campus que mostraba las acciones a realizar al listar los cursos o usuarios con texto, el nuevo campus se centra en mostrar estas acciones con iconos intuitivos. Figura 9.

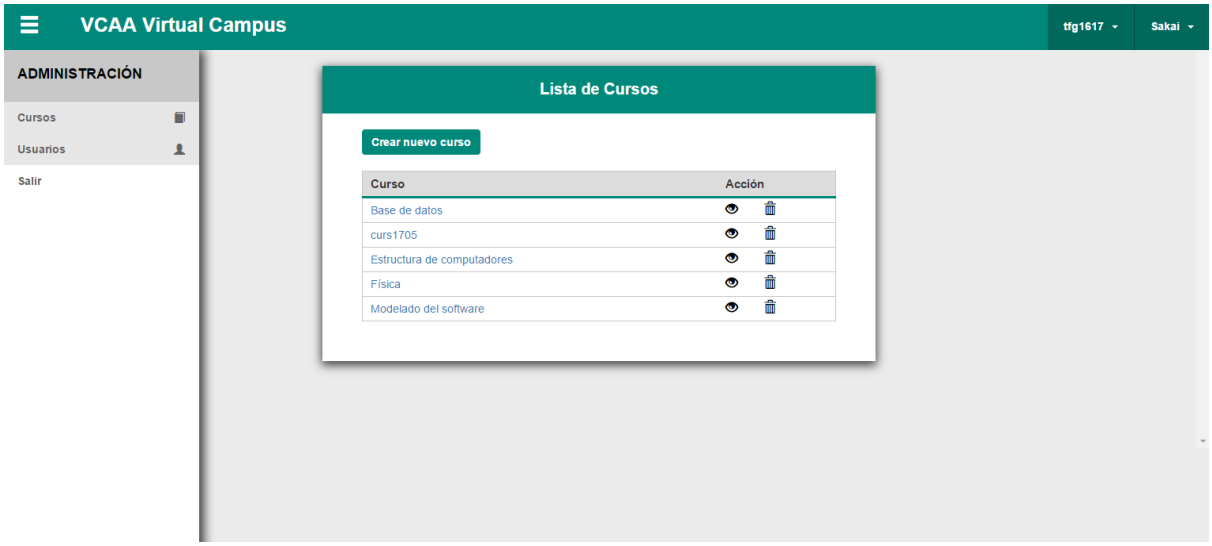

Fig 9.Ventana de curso.

Cuando se acceda a un curso, el sidebar mostrará unas opciones u otras dependiendo del papel que tenga el usuario actual en el curso seleccionado. Es decir, si el usuario es administrador o profesor del curso la aplicación mostrará la opción de Administrar. Figura 10.

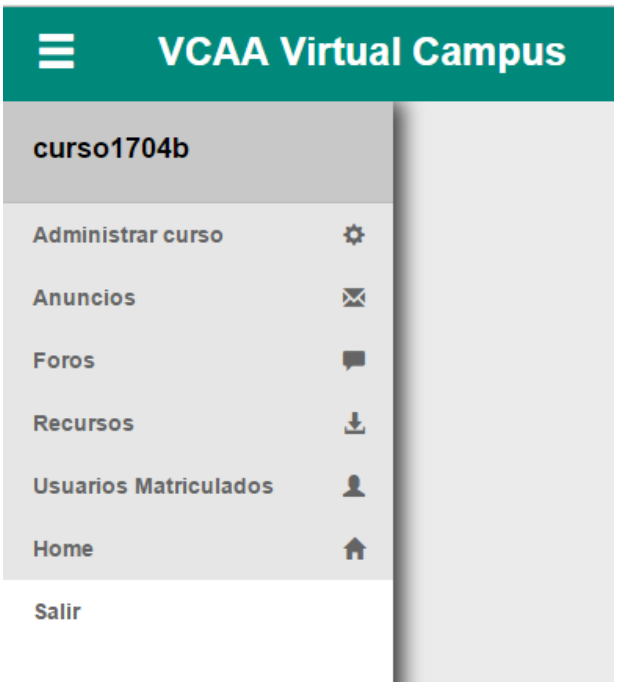

Fig 10.Sidebar de administración de cursos

Cuando se accede a la función de Administrar Curso, se mostrarán los usuarios matriculados en la asignatura, así como el rol que ostentan en el curso, la posibilidad de desmatricular un usuario (distinto del actual) además de la posibilidad de cambiar el rol de los usuarios matriculados y matricular nuevos usuarios en el curso. Figura 11.

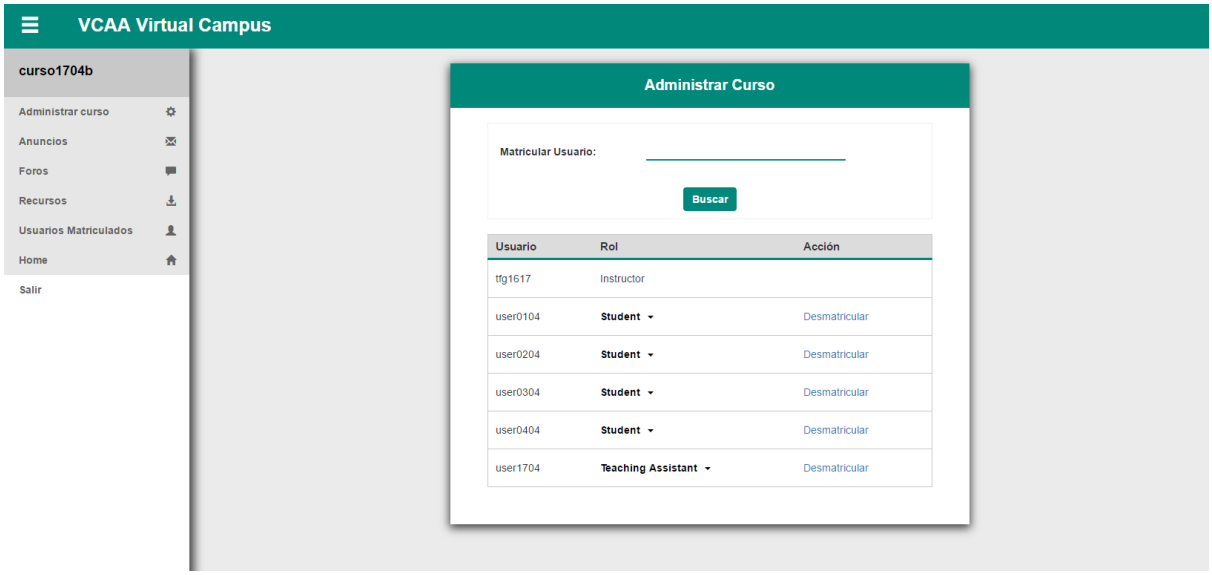

Fig 11. Matriculación de usuarios.

Si se accede a la sección de anuncios del curso, se muestran todos los anuncios publicados hasta la fecha, además de las opciones de borrar anuncios, y de crear anuncios nuevos, si el usuario actual tiene el rol de Profesor en el curso. Figura 12.

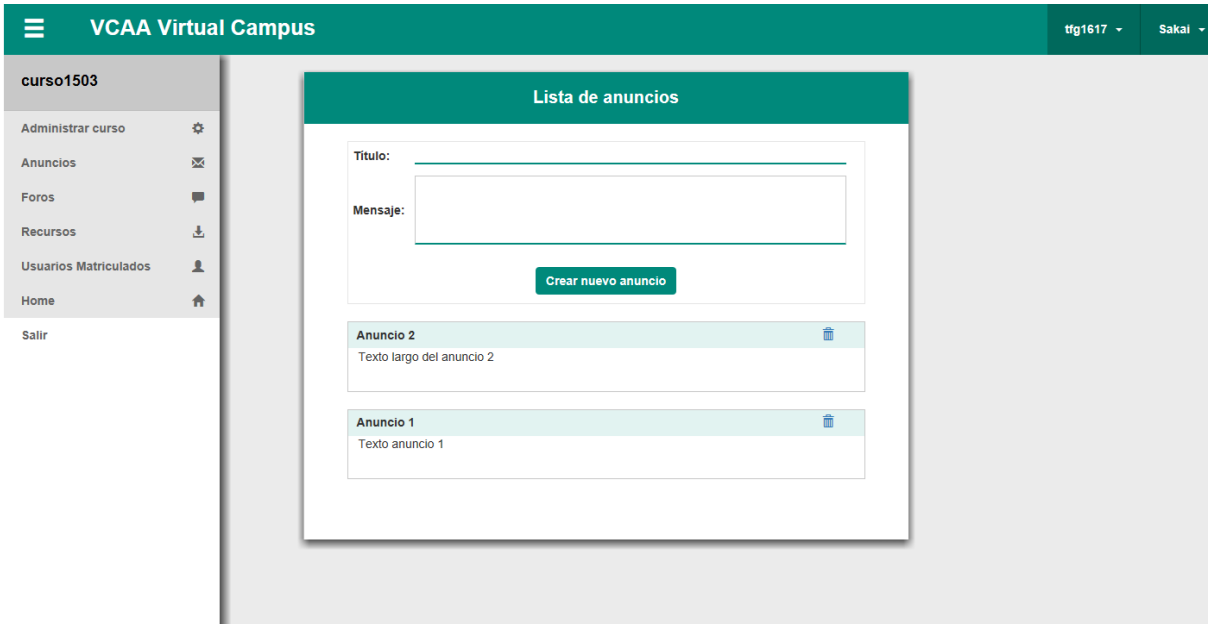

Fig 12. Gestor de anuncios.

La sección de Foros muestra inicialmente los foros que hay abiertos en el curso, con su fecha de creación y su autor. Si el usuario actual es Profesor del curso se mostrarán además la opción de eliminar tema. Figura 13.

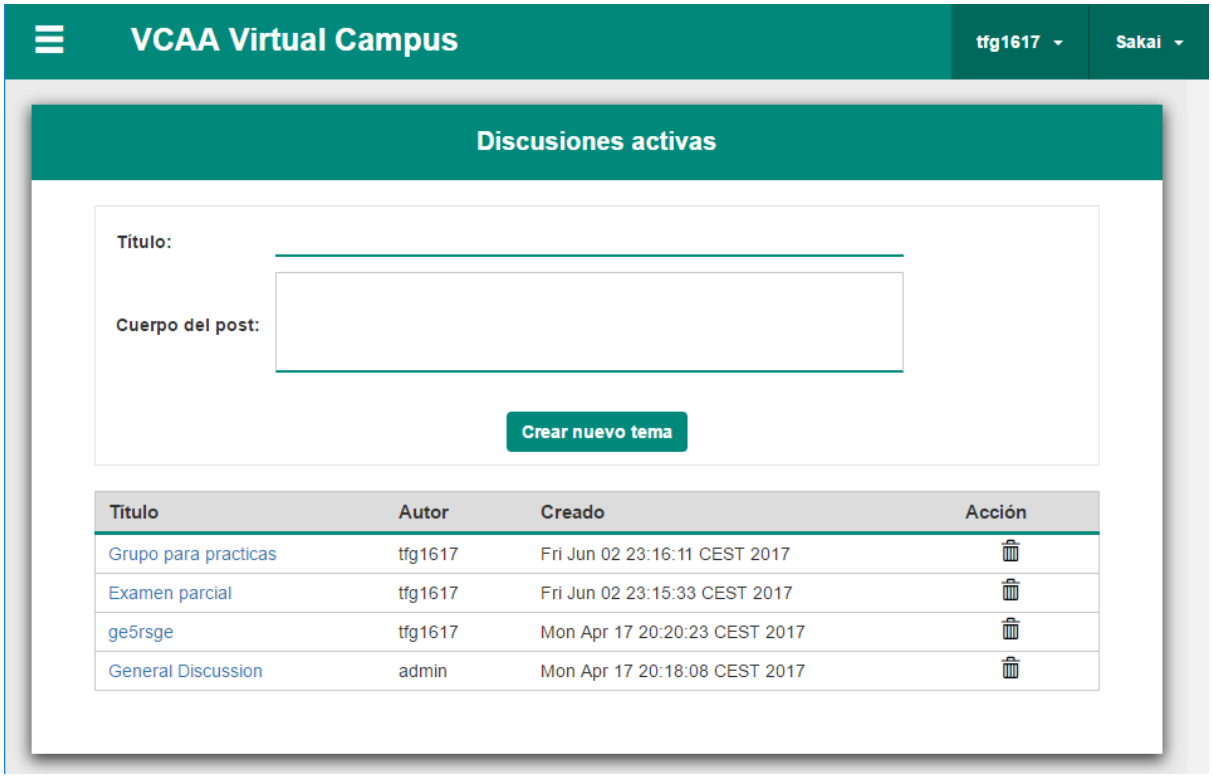

Fig 13. Gestor de foros.

Si se accede a un tema, se mostrará el post principal del tema además de los post que hayan creado otros usuarios en el tema. Figura 14.

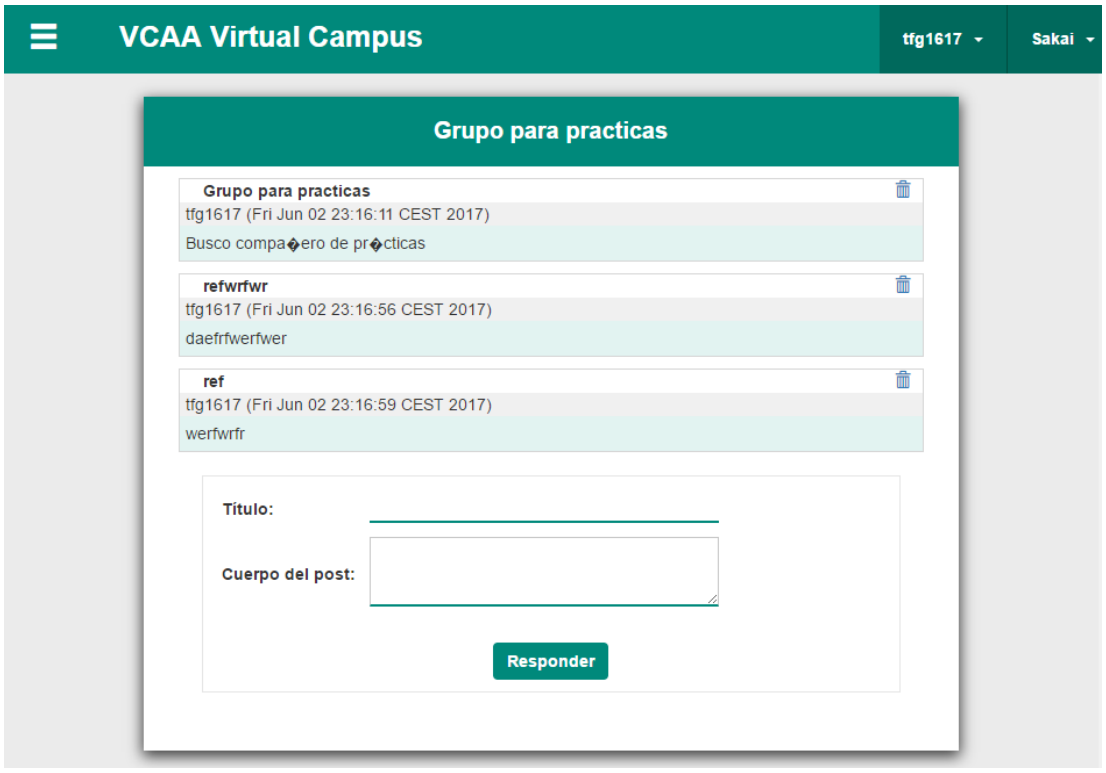

Fig 14. Lista de post de un tema.

Al acceder a la opción de Recursos se desplegará una ventana con los directorios y los recursos subidos al directorio raíz del curso. Si el usuario actual tiene rol de Profesor en el curso podrá crear recursos y directorios nuevos, así como la función de eliminar cualquier recurso o directorio. Figura 15.

Fig 15. Gestión de recursos.

La opción Usuarios Matriculados lista los usuarios matriculados en el curso actual mostrando el rol que tienen en el curso.

Si en el menú de Administración se selecciona la opción Usuarios, se accede a la página de gestión de usuarios, donde se puede crear nuevos usuarios, actualizarlos y borrarlos. Las opciones de visitar, actualizar y borrar vienen representadas por iconos. Figura 16.

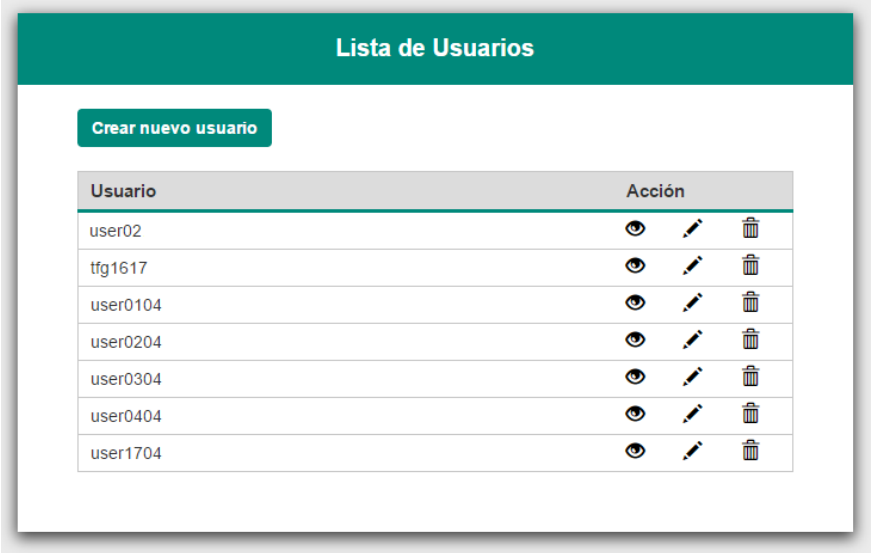

Fig 16.Ventana de listado de usuarios

Cada una de estas opciones abre una página específica en la que realizar la acción deseada. Figuras 17 y 18.

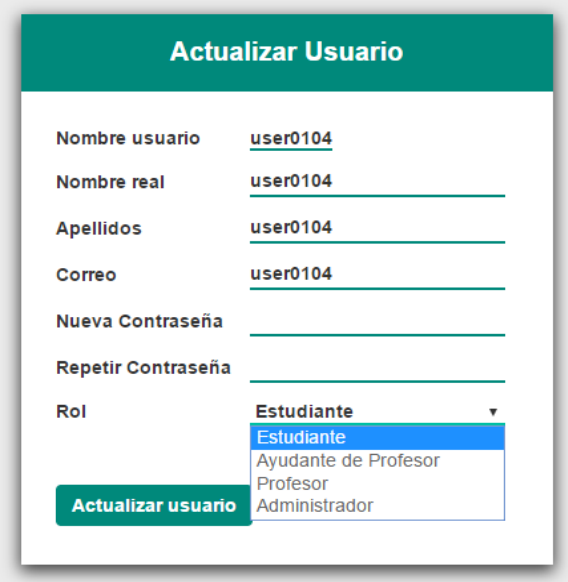

Fig 17, Formulario de creación de usuario

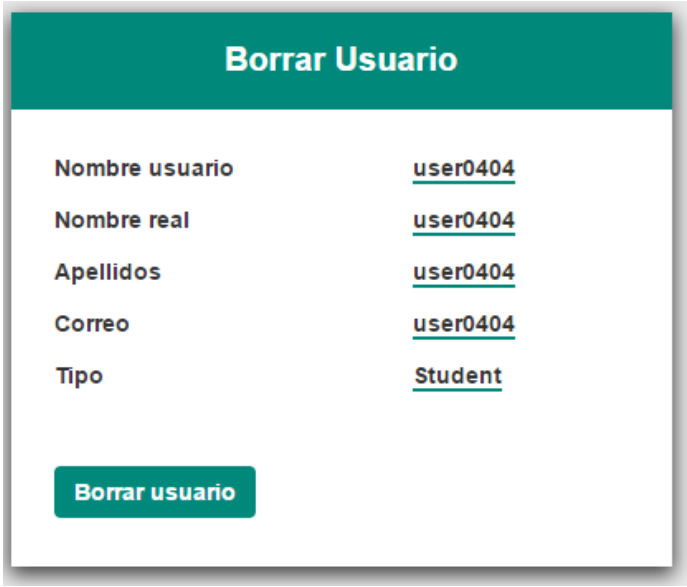

Fig 18.Ventana de confirmación de borrado de usuario

3.2 Análisis de los patrones software utilizados en el proyecto

Hemos utilizado una serie de patrones arquitectónicos que hemos aprendido a lo largo del grado en asignaturas como *Ingeniería del Sofwtare* o *Modelado del Software*, ambas impartidas por nuestro tutor. Tras un análisis sobre los requisitos de este proyecto hemos llegado a la conclusión de que se trataba de un proyecto de reingeniería, ya que había que reconstruir toda la estructura de la aplicación tanto en la parte de la interfaz gráfica como en la parte funcional de negocio. Por la razón anterior hemos decidido aplicar patrones software arquitectónicos que permitan una reingeniería más sencilla y escalable que la que hemos tenido que llevar nosotros a cabo.

Los principales patrones utilizados son aquellos que facilitan una arquitectura multicapa. Esto simplifica futuras reingenierías y escalabilidades de la aplicación porque las capas no están acopladas entre sí. A continuación vamos a describir los patrones que hemos usado en este proyecto.

La aplicación se divide en dos capas, la capa de presentación y la capa de negocio. La capa de presentación está constituida por páginas *JSF*, ficheros XML (*web.xml* y *faces-config.xml)*, los *managed beans* y la parte de seguridad *JAAS*. La capa de negocio consta de las

clases Java y de las invocaciones a los servicios web. En este proyecto no hemos sido responsables del desarrollo de la capa de integración porque hemos decidido usar un framework de mapeado objeto-relacional, en nuestro caso *JPA*.

Otra forma de describir las dos capas bien distinguidas mencionadas anteriormente, es decir que esta aplicación usa el modelo de Cliente-Servidor. El cliente en este proyecto es el navegador web que acceda a nuestra aplicación, y el servidor es nuestra aplicación en sí.

Hemos usado el patrón *Transfer* (Alur et al., 2003) para el transporte de datos entre la capa de presentación y de negocio. Este patrón desacopla ambas capas gracias a sus métodos que garantizan el acceso y la manipulación de sus atributos. Esto quiere decir que ambas capas están acopladas al objeto *Transfer* pero realmente si cambiamos una de ellas, no debe afectar al *Transfer* en cuestión. En esta aplicación se dispone de un *Transfer* por cada entidad que vamos a persistir ya sea en nuestra base de datos o en las bases de datos de las distintas plataformas. Hemos usado este patrón ya que es uno de los básicos para transportar la información entre capas de manera que ambas capas se desacoplen y esto simplifica futuros arreglos escalables de la aplicación.

También hemos utilizado *JavaBeans* JSF, que se encargan de recoger objetos y encapsularlos imitando a un *Transfer* pero esta vez con una mayor funcionalidad. En nuestro proyecto lo hemos utilizado en la capa de presentación para encapsular y guardar toda la información que nos llega desde la parte cliente de nuestra aplicación. También se usan para enviar los datos a la vista de cliente.

Los patrones *Front* y *Application Controller* (Alur et al., 2003) sirven para manejar distintas peticiones en una única clase y que está invoque a negocio dependiendo del tipo invocación que haya recibido. Estos patrones simplifican mucho el acoplamiento entre las dos capas existentes, sobre todo, teniendo en cuenta su implementación JSF. Esto supone una reducción considerable del acoplamiento entre ambas capas y promueve la modularidad de las mismas. En esta aplicación este patrón aparece en la capa de presentación debido a que los *JavaBeans* son los que manejan las peticiones que se hacen desde la parte del cliente y estos invocan a negocio dependiendo de qué tipo de petición le haya llegado. Parte de este patrón es el *Dispatcher*, que se utiliza cuando se desea que una vista maneje una petición y genere una respuesta, gestionando una cantidad limitada de procesamiento de negocio. En nuestro proyecto lo usamos en el fichero *faces-config.xml*, durante la definición de las reglas de navegación para poder redirigir a un determinada página, esta redirección se lleva a cabo dependiendo del *outcome* que esté definido en el *JavaBean*.

En negocio hemos usado *Business Object* (Alur et al., 2003) implementados como entidades JPA. Este patrón organiza el código de negocio en unidades lógicas que facilitan la mantenibilidad del sistema. Estas unidades lógicas representan los objetos de la realidad que vamos a persistir en la base de datos o en las distintas plataformas. En nuestro proyecto tenemos definidas como *Bussines Object* las entidades que vamos a persistir en nuestra base de datos, es decir, tres entidades de las cinco analizadas.

Hemos usado el patrón *Application Service* (Alur et al., 2003) con el objetivo de centralizar la lógica de negocio a través de distintos componentes de la capa de negocio y servicios. Además este patrón se encarga de definir las reglas de negocio necesarias sobre los objetos que más adelante vamos a persistir. Otro motivo por el que lo usamos en nuestra aplicación es porque elimina la lógica presente en la capa de presentación. En nuestro proyecto existe un *Application Service* para cada módulo de nuestro análisis (estos módulos los definimos más adelante de una manera más detallada). En cada uno de ellos se recopilan las operaciones *CRUD* ya que es la funcionalidad básica que hemos decidido que tengan. Este patrón junto con los patrones *Controller*, *Transfer* y *Abstract Factory* es de los patrones más importantes en este proyecto de reingeniería ya que son los patrones que facilitan futuras versiones escalables que proporcionen mejoras.

El patrón *Abstract Factory* (Gamma et al, 1994) es el encargado de construir objetos de una misma familia, esto simplifica la creación de objetos que son similares pero sin ser iguales, ya que podemos tener distintos *Application Service* para cada uno de nuestros módulos. Estos *Application Service* son de la misma familia, pero cada uno gestiona una entidad distinta. En nuestro proyecto hemos usado este patrón para realizar las instancias de los *Application Service* de cada uno de los módulos.

En la capa de negocio hemos usado el patrón *Singleton* (Gamma 1994). Este patrón permite que la clase cree una instancia de sí misma. Con esto nos aseguramos que la clase sólo se instancie una única vez ya que las clases *Singleton* de nuestro proyecto son elementos críticos que de tener distintas instancias provocaría pérdida de información. En esta aplicación este patrón aparece representado en la capa de negocio, ya que es donde la lógica de nuestra aplicación es más sensible y está mejor protegida.

3.3 Modelo del dominio

Después de un análisis exhaustivo llegamos a la conclusión de que todas las entidades

que habíamos detectado no iban a ser persistidas en nuestra base de datos a pesar de que en nuestra aplicación se pueden tener acceso a los recursos, foros o anuncios, estos deberían ser controlados a través de servicios web. Estas últimas entidades están gestionadas internamente por la plataforma en la que desplegamos. Figura 19.

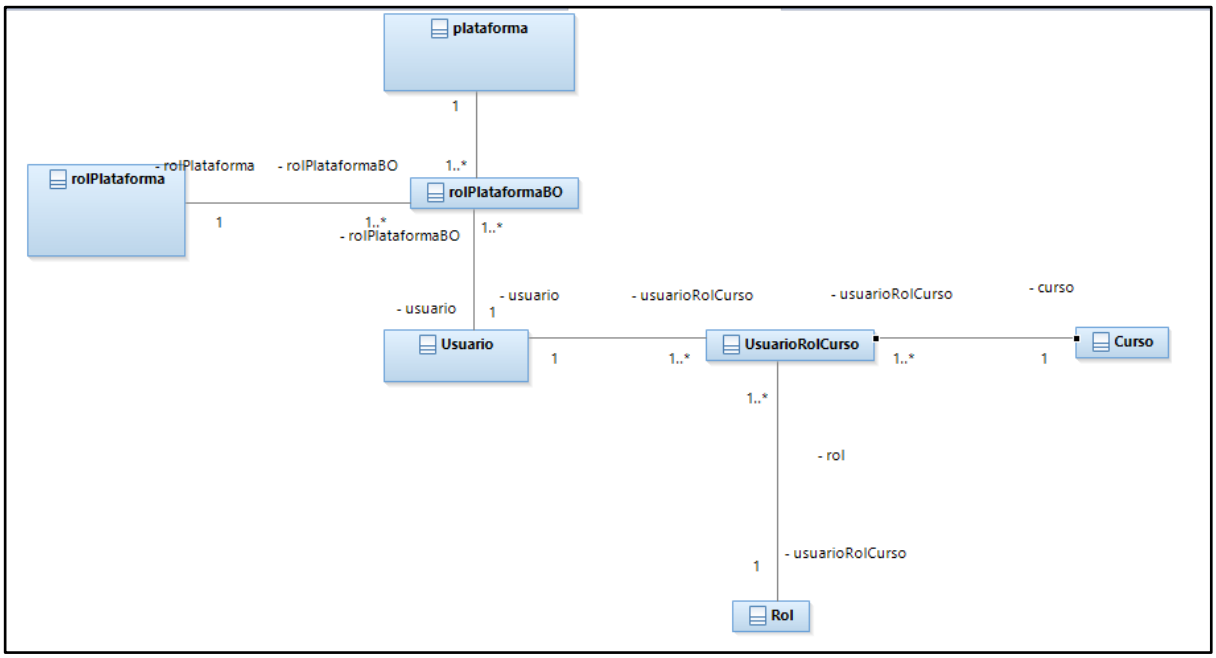

Fig 19.Modelo del dominio.

Como podemos observar en la figura anterior, hemos decidido persistir tres de las cinco entidades analizadas. Estas entidades tenían una relación n-aria entre ellas, por lo que a la hora de representarlo en nuestra base de datos hemos tenido que crear clases intermedias para simplificar el desarrollo. Estas clases nos sirven en nuestra base de datos como claves compuestas ajenas, ya que en ciertas clases no es suficiente con una columna identificativa. Al meter una clase intermedia, las relaciones entre las entidades y la nueva clase pasan a ser relaciones 1 a n.

3.4 Análisis de los requisitos

A continuación vamos a describir de una manera técnica los requisitos funcionales del módulo de usuario. Hemos optado por tomar este módulo como ejemplo debido a que la funcionalidad básica de los demás módulos es similar a esta, y la inclusión de todos los requisitos de la aplicación aumentaría de forma significativa el tamaño de la memoria. El formato elegido para representar de una manera técnica estas funcionalidades es el que se recoge en el IEEE Std. 830-1998 (IEEE, 1998) para las definiciones de los requisitos de usuario y para las definiciones de requisitos de sistema de una aplicación software.

3.4.1 Requisitos de usuario

En nuestra aplicación disponemos de cinco módulos. Esta sección describe los requisitos de usuario de los cinco módulos, incluyendo los datos necesarios a la hora de realizar las invocaciones de ciertas operaciones

Gestión de usuarios

- *Alta de usuario (username, nombre, apellidos, contraseña, correo, rol, id plataforma): resultado entero*

  En la operación alta de usuario necesitamos estos parámetros, ya que el *username* es el atributo identificativo del usuario. La *contraseña* es necesaria en este caso dado que la aplicación consta de un sistema de *login* y se necesita para validar la cuenta. El *correo* es necesario, ya que va a ser el método mediante el cual el profesor u otros alumnos van a poder comunicarse entre sí. El *rol* también será necesario dado que es lo que define dentro de esta plataforma a qué páginas se tiene acceso. El *nombre* y *apellidos* no son obligatorios, dado que no es una información relevante en este caso aunque sí convendría rellenarlos. El *id plataforma* es siempre necesario para saber en cuál de ellas queremos dar de alta al usuario.

- *Baja de usuario (username, id plataforma): resultado entero*

  Al pulsar sobre el botón de borrado (icono de papelera), se mostrará un formulario con los datos del usuario y un botón de confirmar.

- *Mostrar usuario (username, id plataforma): id usuario, username, nombre, apellidos, correo, rol en la plataforma*

  Al pulsar sobre el botón de mostrar (icono de un ojo), se mostrarán los datos del usuario en un formulario.

- *Actualizar usuario: (username, nombre, apellidos, contraseña, correo, rol, id plataforma): resultado entero*

  Al pulsar sobre el botón de actualizar (icono de un lápiz) se mostrarán los datos del usuario en un formulario. En este formulario el *username* no se puede modificar, ya que es el atributo identificativo de la clase, si este atributo se pudiese modificar podrían producirse inconsistencia de datos en nuestra base de datos.

- *Mostrar todos los usuarios (id plataforma): Lista de usuarios con id usuario, username*

Una vez hagamos click en el sidebar, en el apartado de Administración de usuarios, se nos mostrará un listado de todos los usuarios registrados en la plataforma.

Gestión de Cursos

- *Alta Curso (nombre completo, nombre corto, categoría, descripción, usuario instructor, id plataforma): resultado entero*

  En la operación alta de usuario necesitamos seis parámetros. El *nombre corto* se usado como clave única para diferenciarlo de otros cursos, entendidos cursos como asignaturas. El campo *categoría* se usa para identificar la pertenencia a un grupo de asignaturas pertenecientes al mismo grado. La *descripción* se usa para mostrar un breve resumen sobre el compendio de dicha asignatura. El *usuario instructor* se utiliza para que al crearse el curso tenga ya asignado un usuario con el rol de profesor, si se manda vacío matricula al usuario actual como profesor del curso creado.

- *Matricular usuario (nombre usuario, rol, id curso, id plataforma): resultado entero*

  Esta operación matricula un usuario en el curso con el rol introducido.

- *Eliminar Curso (id curso, id plataforma): resultado entero*

  Al pulsar sobre el botón de borrado (icono de papelera) se mostrará un formulario con los datos del curso seleccionado y un botón para confirmar el borrado.

- *Mostrar curso (nombre curso, id curso, id plataforma): id curso, nombre curso, categoría, nombre largo, categoría, descripción*

  Tras haber pulsado el botón de mostrar curso (símbolo de un ojo) se nos muestra un formulario con los datos del curso.

- *Mostrar todos los cursos (id plataforma): Lista cursos con nombre curso, id curso*

  Muestra la lista de los cursos dados de alta en la plataforma.

Gestión de anuncios

- *Alta de anuncio (id curso, título anuncio, contenido mensaje, id plataforma): resultado entero.*

  Da de alta a un anuncio en un curso seleccionado anteriormente.

- *Mostrar anuncios (id curso, id plataforma): Lista de anuncios con título, contenido de anuncio, id de anuncio*

En esta operación se listan todos los anuncios de una determinada asignatura previamente seleccionada ordenados por fecha de creación, siendo antes visibles los anuncios más recientes.

- *Eliminar anuncio (id anuncio, id curso, id plataforma): resultado entero*
  Desde la ventana en la que aparecen listados todos los anuncios de una asignatura, seleccionamos el botón de borrar (icono de papelera) y se elimina el anuncio deseado. La lista de anuncios mostrados se actualiza al momento.

Gestión de foros

- *Alta de tema (título tema, contenido tema, id curso, id plataforma): resultado entero*
  En esta operación es obligatorio introducir el *título del tema* ya que aunque no sea el atributo identificativo en las plataformas, no sería adecuado permitir la creación de temas con el *título del tema* vacío.

- *Responder post (id tema, título tema, contenido tema, id curso, id plataforma): resultado entero*
  Dentro de un tema podemos responder a los post. Esta operación permite crear *post* dentro de un tema. El *cuerpo del mensaje* es obligatorio porque no sería lógico permitir respuestas que no contengan nada.

- *Borrar post (id post, id tema, id curso, id plataforma): resultado entero*
  Esta funcionalidad solo está disponible para los administradores de dicho curso. Veremos que aparece un botón a la derecha de los *post* (icono de papelera) que al pulsarlo, borrará dicho post.

- *Borrar tema (id tema, id curso, id plataforma): resultado entero*
  Esta operación se invoca mediante un botón (icono de papelera) colocado a un lado del nombre del tema. Una vez pulsemos el botón el *tema* y los post que tenga serán borrados.

- *Mostrar tema (id tema, id curso. id plataforma): id tema, título tema, lista de posts publicados en el tema*
  Esta operación está incluida en el propio nombre del tema, dentro del formulario. Si hacemos click sobre el nombre del tema se nos redirigirá a una página donde se nos muestran todos los mensajes que tiene ese tema.

- *Mostrar todos los temas (id curso, id plataforma): Lista de temas con id tema, título tema, id curso, autor, fecha de creación y lista de post del tema*

  Esta operación es invocada mediante el panel de administración. Una vez hacemos click sobre *Administrar Foros* se nos redirige a un formulario que contiene la lista de todos los temas creados dentro del curso que estemos administrando.

Gestión de Recursos

- *Alta carpeta (id curso, id usuario, nombre carpeta, ruta): resultado entero*

  Aparecerá un campo de texto en el que debemos introducir el nombre que le vamos a dar a la nueva carpeta. Al pulsar en el botón *Crear carpeta* se creará una carpeta en la ruta actual.

- *Alta de archivo (id curso, id usuario, fichero, ruta fichero): resultado entero*

  Aparecerá un selector de archivos, buscamos el archivo que queramos subir y pulsamos el botón subir fichero. Por defecto estas dos acciones provocan que tanto las carpetas o los archivos nuevos se creen en el directorio raíz, para evitar esto es necesario recurrir a la funcionalidad mostrar recurso, que se explicara en los siguientes apartados,

- *Mostrar todos los recursos (id curso, ruta, id plataforma)*

  Para poder todos los recursos de una asignatura únicamente se debe pulsar el botón recursos del desplegable que se encuentra a la izquierda. Por defecto nos muestra todos los ficheros y carpetas que se encuentran en el directorio raíz.

- *Mostrar archivo (id curso, id recurso)*

  Al pulsar el botón de mostrar (icono de ojo) se muestra el nombre del fichero, la ruta en la que se encuentra, el tamaño del archivo, la fecha de subida y el botón de *Descargar* en un formulario.

- *Mostrar carpeta (id curso, id recurso)*

  Al pulsar sobre el nombre de la carpeta o sobre el botón de mostrar (icono de ojo) se mostrarán los recursos de dicha carpeta. Esto además nos permite subir ficheros o crear carpetas dentro de esta para clasificar mejor los recursos.

- *Eliminar recurso (id recurso, id curso): resultado entero*

  Al hacer click sobre el botón de eliminar (icono de papelera) se elimina el fichero o carpeta (y todo lo que haya dentro de ella) seleccionado.

- *Descargar recurso (id recurso, id curso): resultado entero*
  Para realizarla, desde el listado que muestra todos los recursos, se debe seleccionar el botón de descarga de archivos.

3.4.2 Requisitos de sistema

En este apartado vamos a introducir los requisitos anteriormente definidos a nivel de usuario, pero esta vez a un nivel más detallado. A continuación describiremos únicamente el módulo de usuario, como muestra del resto de requisitos. En la plantilla a rellenar se definen varios campos. El campo de *prioridad* indica la relevancia y como su nombre indica la prioridad para implementar esta función, las funciones con una prioridad más alta deberán ser atendidas en un plazo de tiempo más corto. En *estabilidad* se indica la necesidad de que está función tiene que estar implementada de tal manera que no sea necesario modificarla en un futuro. *Descripción* contiene una breve descripción de la funcionalidad del método. En *entrada* se define la entrada del sistema, es decir, que va a recibir nuestra función. En *salida* declaramos que ha pasado en el sistema tras darse una ejecución correcta. En *origen* se especifica de dónde proceden los datos que se le suministran a la función. *Destino* define el lugar donde se va a ejecutar la función. El campo *necesita* son los datos que necesita la función y que no son datos de entrada de la función, así como otros elementos operativos necesarios. Además en los campos *precondición* y *postcondición* se definen condiciones lógicas que son necesarias antes de que se ejecute la función y después de ejecutarse. Por último en *efectos colaterales* definimos, si procede, algo de información que sea de interés tras haber ejecutado la función.

Nota: Dependiendo de la plataforma que hayamos desplegado, el atributo identificativo de la entidad usuario irá variando entre id o username.

<u>Función: Alta usuario</u>
Prioridad: Alta.
Estabilidad: Alta.
Descripción: Añade un nuevo usuario a la plataforma desplegada. Figura 20.
Entrada: username, nombre, apellidos, contraseña, correo rol, id plataforma.
Salida: Nuevo usuario creado en la plataforma con el rol asignado.
Origen: Los datos se introducen mediante teclado en la página cliente de alta de usuario.
Destino: Sistema.
Necesita: Base de datos, conexión con la plataforma desplegada mediante *internet* y conexión a *Internet*.

Precondición: No hay usernames repetidos. Un usuario sólo no puede estar más de una vez en la misma plataforma, solamente puede crear usuarios un usuario con rol de administrador o profesor.

Postcondición: Alta del usuario con username de entrada distinto al username de los usuarios existentes.

Efectos laterales: Se podrán registrar usuarios con el mismo username pero en distintas plataformas, simulando así el hecho de estar registrado con la misma cuenta en las distintas plataformas.

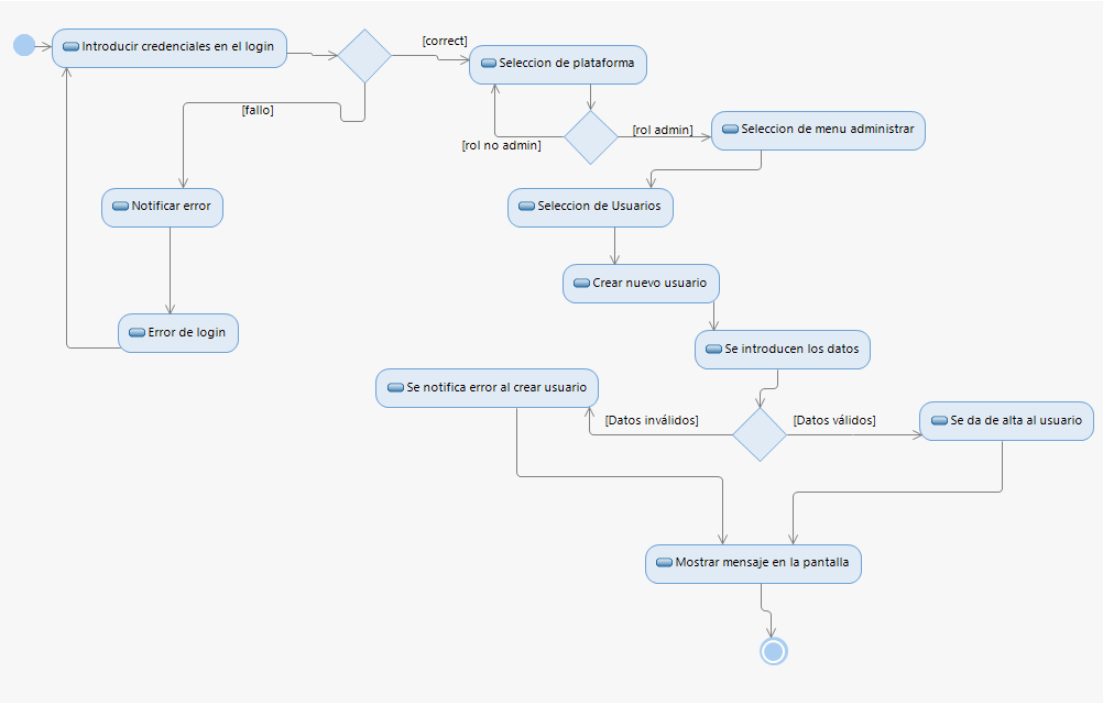

Fig 20. Diagrama de actividad de alta de usuario.

Función: Actualiza usuario

Prioridad: Alta.

Estabilidad: Alta.

Descripción: Actualiza un usuario en la plataforma desplegada. Figura 21.

Entrada: *username/id**, nombre, apellidos, contraseña, correo, rol, id plataforma.

Salida: Usuario actualizado en la plataforma desplegada.

Origen: Los datos se introducen mediante teclado en la página cliente de actualización de usuario.

Destino: Sistema.

Necesita: Base de datos, conexión con la plataforma desplegada mediante *internet* y conexión a *Internet*.

Precondición: No hay usernames repetidos. Un usuario sólo no puede estar más de una vez en la misma plataforma.

Postcondición: Datos en formato válido para la plataforma. Si existe, usuario actualizado en la base de datos y en la plataforma.

Efectos laterales: ---

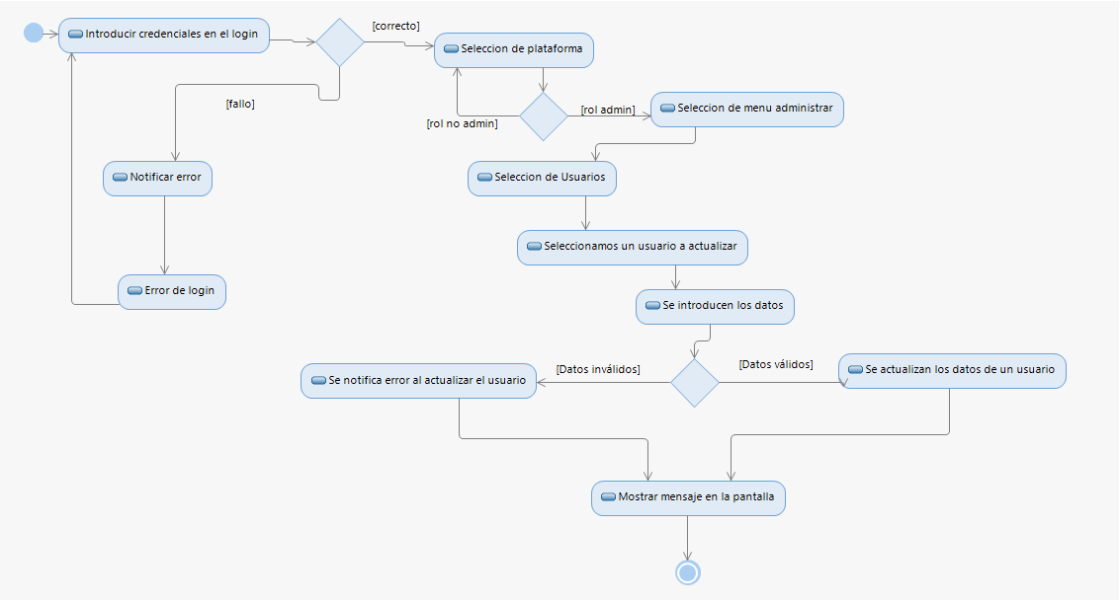

Fig 21. Diagrama de actividad de actualizar usuario.

Función: Mostrar usuario

Prioridad: Alta.

Estabilidad: Alta.

Descripción: Muestra un usuario de la plataforma desplegada. Figura 22.

Entrada: *username/id**, id plataforma.

Salida: Datos del usuario de la plataforma desplegada.

Origen: Esta función se invoca tras pulsar el icono de mostrar (icono de ojo) en la página principal de administrar usuarios.

Destino: Sistema.

Necesita: Base de datos, conexión con la plataforma desplegada mediante *internet* y conexión a *Internet*.

Precondición: No hay usernames repetidos. Un usuario no puede estar más de una vez en la misma plataforma.

Postcondición: No se modifica el estado en la base de datos.

Efectos laterales: ---

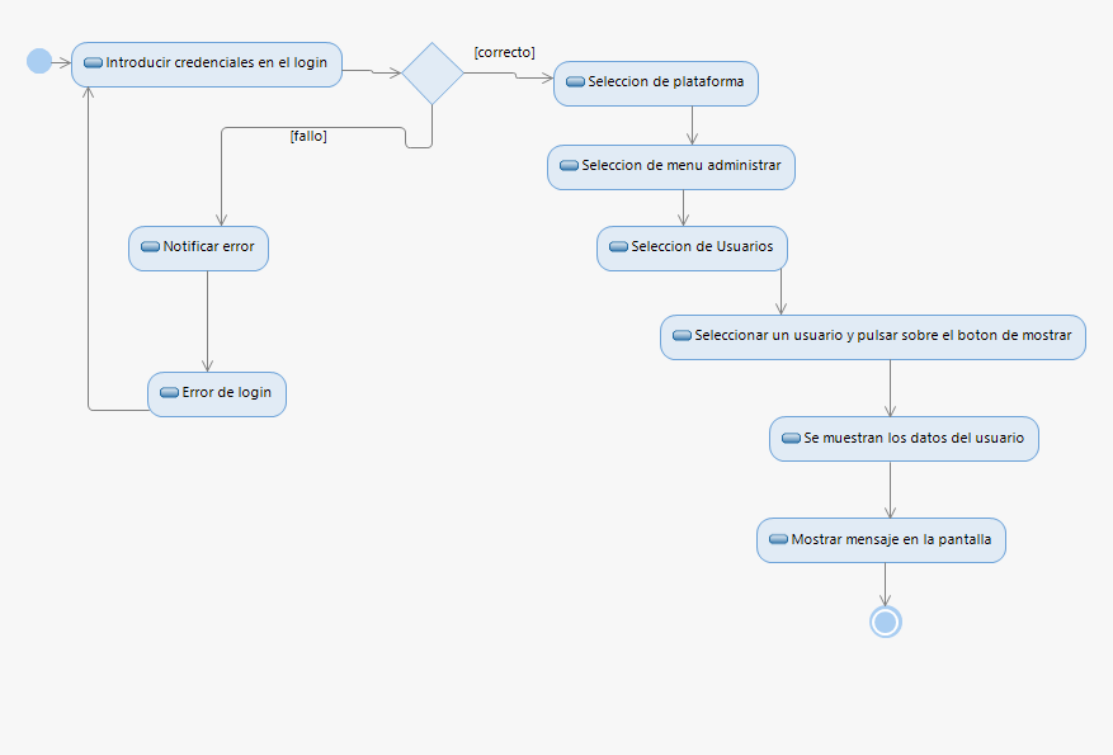

Fig 22. Diagrama de actividad de mostrar usuario.

Función: Borrar usuario

Prioridad: Alta.

Estabilidad: Alta.

Descripción: Borra un usuario en la plataforma desplegada. Figura 23.

Entrada: *username/id**, id plataforma.

Salida: Usuario eliminado de la plataforma desplegada.

Origen: Los datos se introducen mediante teclado en la página cliente de eliminación de usuario.

Destino: Sistema.

Necesita: Base de datos, conexión con la plataforma desplegada mediante *internet* y conexión a *Internet*.

Precondición: No hay usernames repetidos. Un usuario no puede estar más de una vez en la misma plataforma.

Postcondición: Datos en formato válido para la plataforma. Usuario eliminado de la base de datos y en la plataforma.

Efectos laterales: Si el usuario a eliminar está matriculado en algún curso, este es automáticamente eliminado del listado de usuarios matriculados para dicho curso.

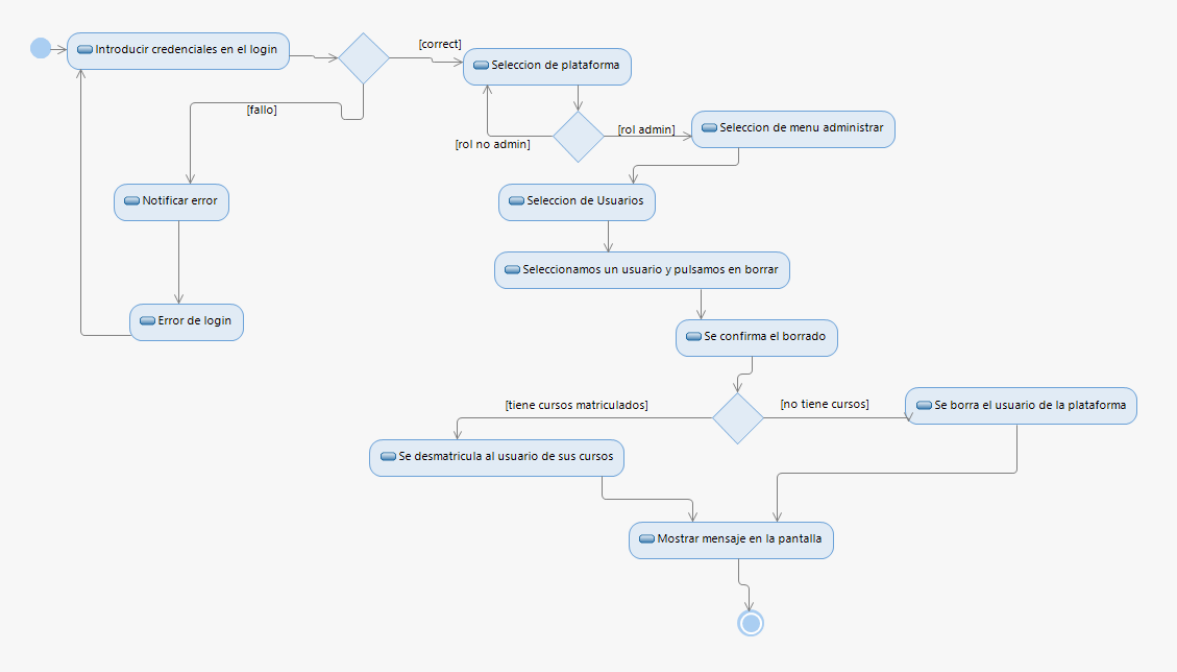

Fig 23. Diagrama de actividad de borrar usuario.

Función: Mostrar todos usuarios

Prioridad: Alta.

Estabilidad: Alta.

Descripción: Muestra todos los usuarios de la plataforma desplegada. Figura 24.

Entrada: id plataforma.

Salida: Lista de usuarios registrados en la plataforma desplegada.

Origen: Esta función se invoca tras pulsar Administrar/Usuarios, en la *Home* de la aplicación.

Destino: Sistema.

Necesita: Base de datos, conexión con la plataforma desplegada mediante *internet* y conexión a *Internet*.

Precondición: No hay usernames repetidos. Un usuario sólo no puede estar más de una vez en la misma plataforma.

Postcondición: ---

Efectos laterales: ---

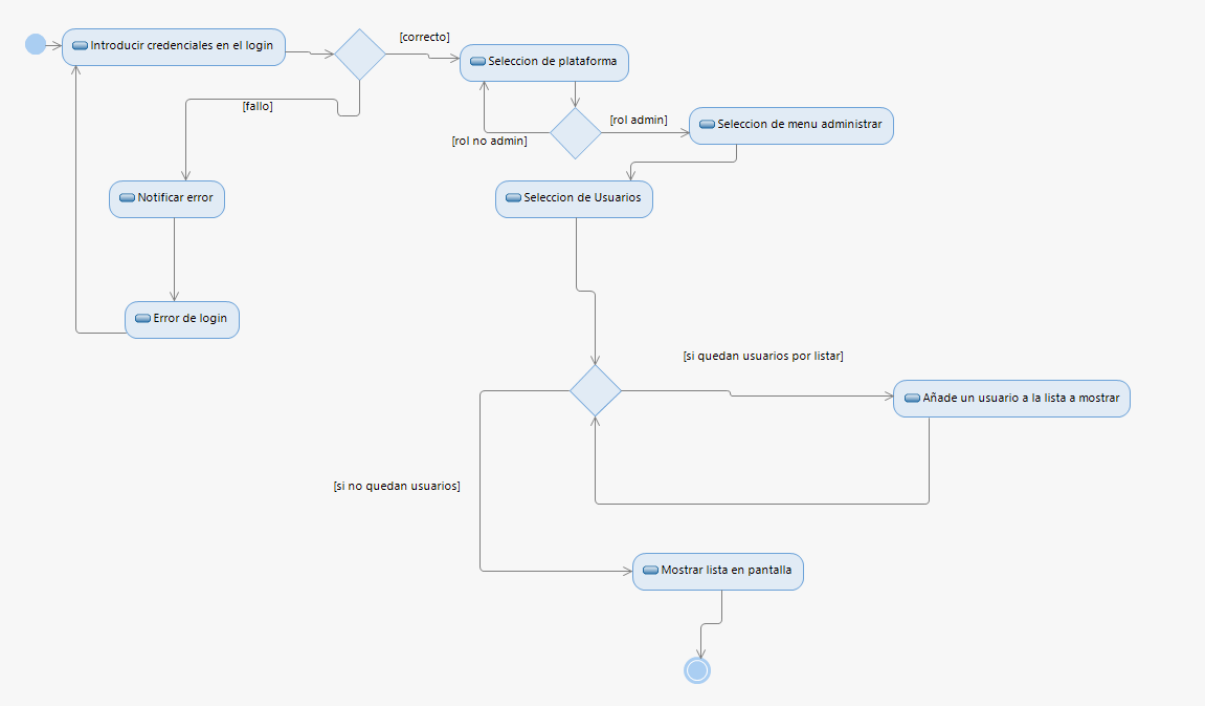

Fig 24. Diagrama de actividad de mostrar todos.

3.5 Diseño de la capa de presentación.

En cuanto a la capa de presentación, a continuación vamos a analizarla detenidamente. En un principio se desarrolló cada página web desde cero, teniendo así un total de veinticinco páginas *JSF* basadas en plantillas. En nuestras plantillas hemos definido un *sidebar* que contiene las acciones que se puedan realizar en la pantalla actual en la que estamos. Por ejemplo, si nos encontramos en la ventana de administración de curso, el sidebar nos presentará todas las acciones que impliquen alguna acción con la entidad curso. Si en lugar de estar en la ventana administración de curso, estamos en la ventana de administración de usuario, el sidebar cambiará y nos presentará las acciones de administración de usuarios. Hemos dado la opción al usuario de poder ocultar este *sidebar* siempre que lo crea necesario. En esta aplicación se dispone de dos tipos de *sidebar*, dependiendo del rol asignado al usuario que ha hecho *login*. En el caso de que seamos un usuario que no sea administrador, solo se nos mostrarán las opciones de perfil y lista de cursos en los que estamos matriculados. En el caso de que poseamos una cuenta de administrador, se nos mostrará un *sidebar* con una funcionalidad de administrar. Una vez accedemos a esa funcionalidad, se nos mostrarán en el *sidebar* todas las opciones disponibles de administración. Cada sección se encarga de administrar una entidad de la plataforma

También definimos como elemento de la plantilla un *header*. Hemos crecido razonable tener siempre la opción de poder cambiar de plataforma rápidamente, y de poder cerrar sesión independientemente de la página en la que nos encontremos en dicho *header*.
Para estructurar los contenidos de cada entidad hemos llegado a la conclusión de que era necesario crear subcarpetas, una por cada entidad. Cada carpeta contiene todos los elementos web relacionados con dicha entidad.

Una vez definida la plantilla que hemos usado, vamos a pasar a hablar del contenido de cada página. Cada una de ellas, aparte de contener un *sidebar* y un *header* contiene un *MainTableContent*. Este *MainTableContent* es un formulario *HTML* con un determinado número de campos. Estos campos están conectados mediante *JavaBeans* con clases *Java* y directamente representan los atributos del *JavaBeans* que se esté usando en ese momento. Para invocar a los métodos de los *JavaBeans* se utiliza un *outcome* que está definido en el botón de envío. Una vez hemos hecho el envío se nos dará retroalimentación en una nueva página aclarando si la operación recién invocada ha sido un éxito.

A continuación mostramos los diagramas de presentación de las páginas *JSF* para todas las funcionalidades que realiza la entidad Usuario, estos diagramas describen la estructura de estas páginas. Estos diagramas han sido creados siguiendo el perfil definido en WAE4JSF (Cortés & Navarro, 2017). Fig 25-2

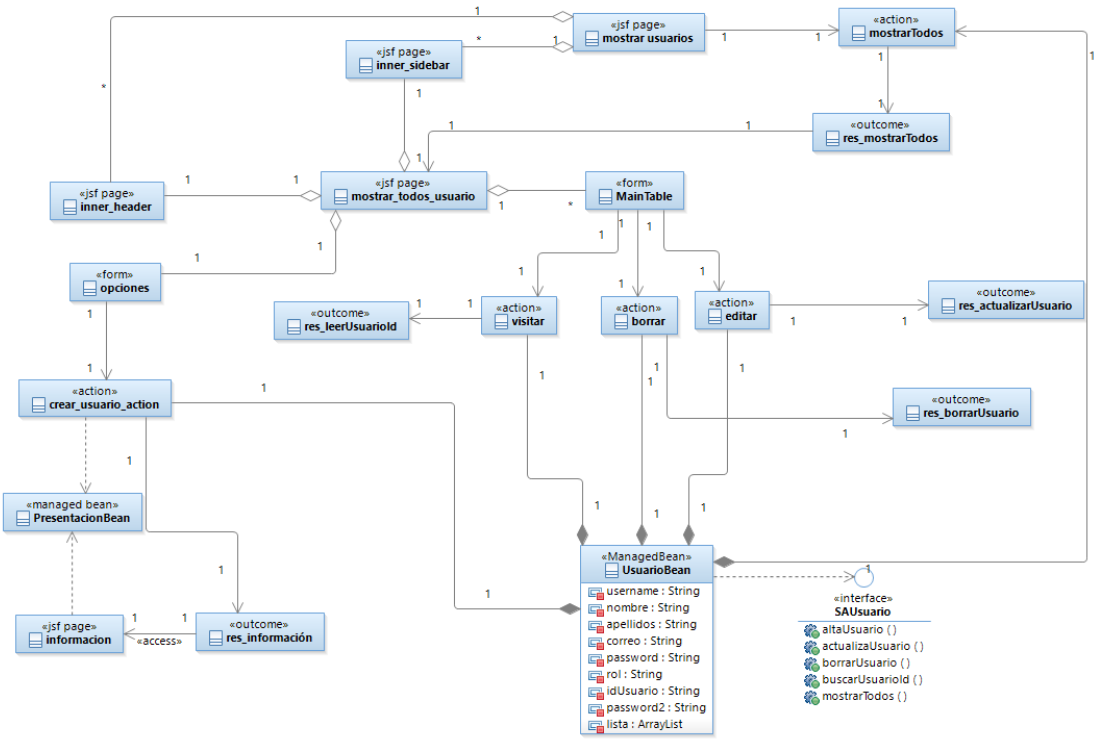

Fig 25. Diagrama de clase de mostrar todos.

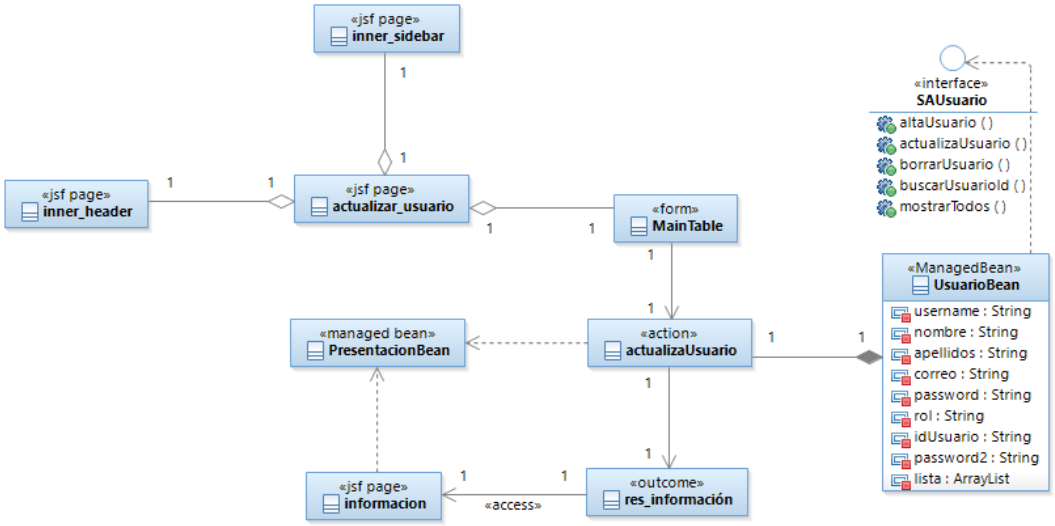

Fig 26. Diagrama de clase de actualizar usuario.

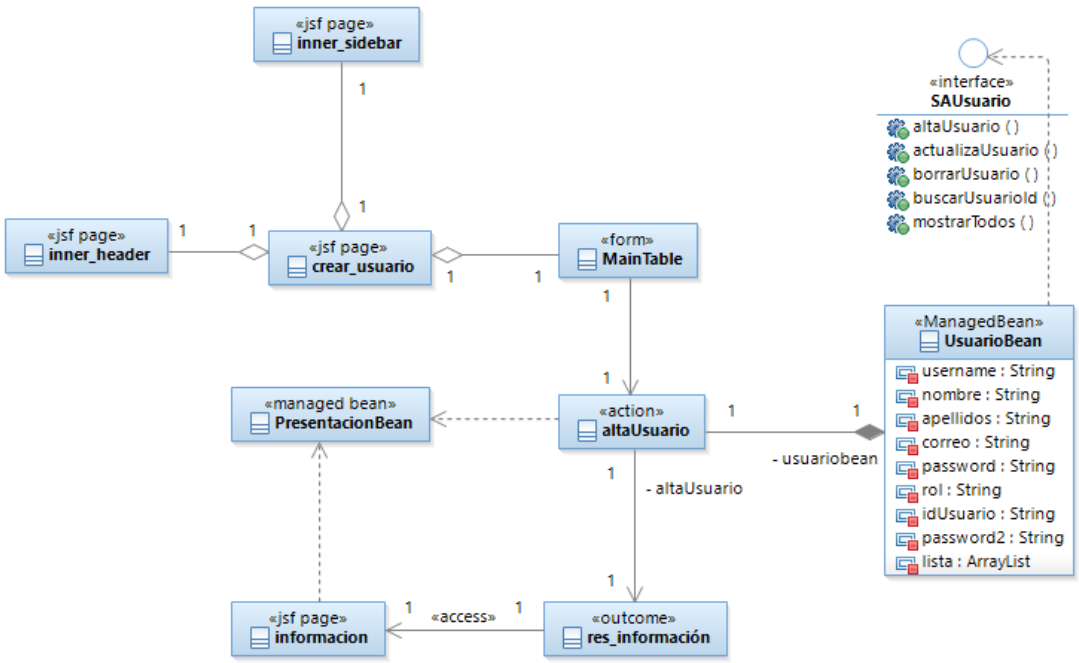

Fig 27. Diagrama de clase de alta usuario.

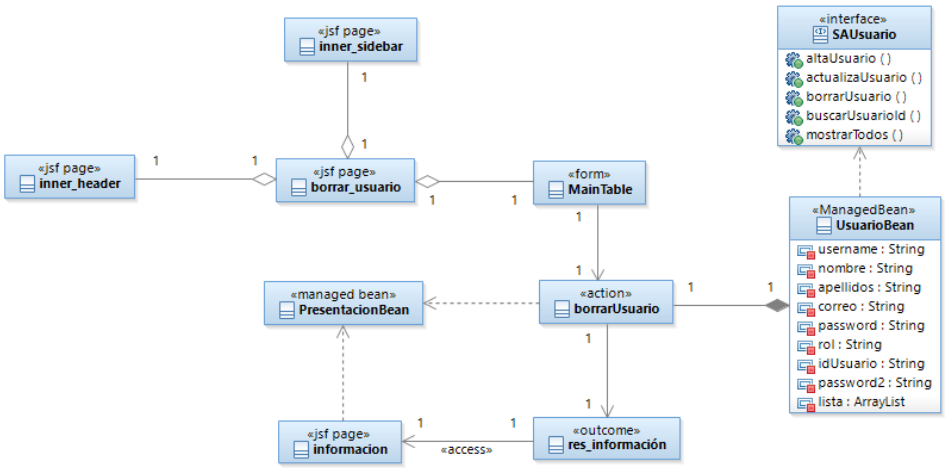

Fig 28. Diagrama de clase de baja usuario.

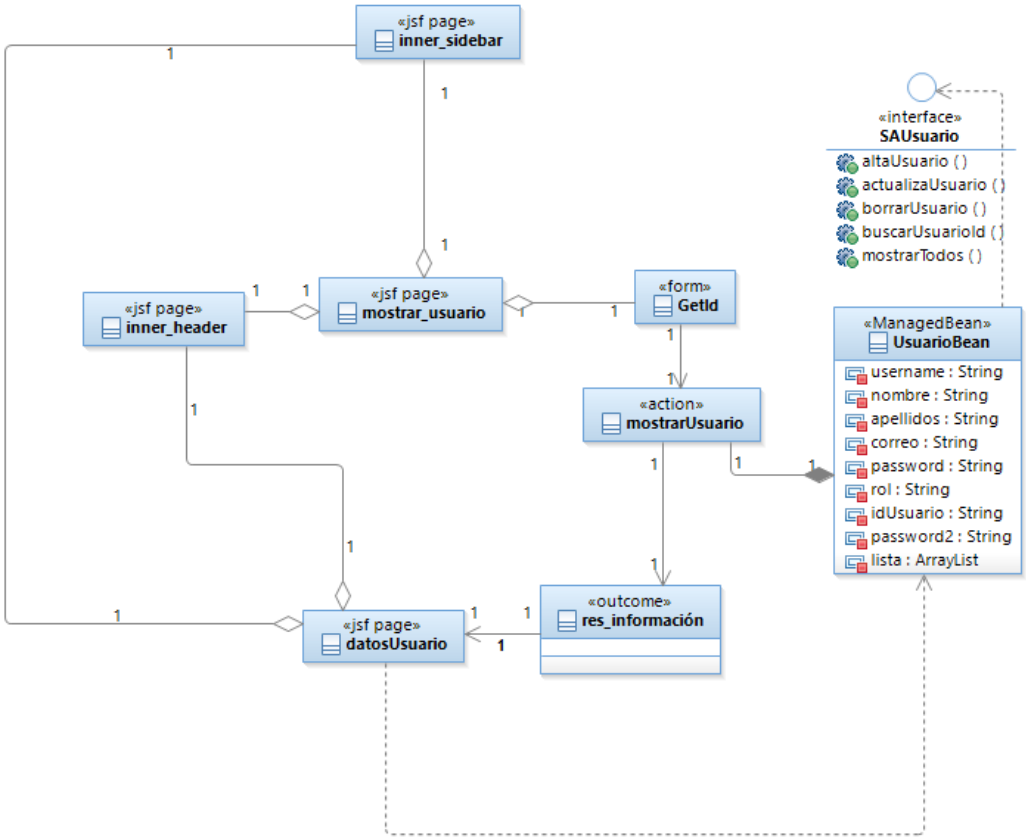

Fig 29. Diagrama de clase de mostrar usuario.

3.6 Diseño de la capa de negocio.

A continuación vamos a describir el diseño de la capa de negocio. Para esto hemos diseñado diagramas de clase y diagramas de secuencia de las operaciones del módulo Usuario. Esta decisión se ha llevado a cabo dada la similitud con los demás módulos en cuanto a estructura se refiere. La estructura de esta capa es modular porque esto facilita los futuros desarrollos y posibles proyectos de mejora. En esta capa contamos con un *Application Service* que se encarga de gestionar todas las reglas de negocio: como puede ser que dos usuarios distintos no compartan el mismo *Username*, o que no se pueda borrar a un usuario que no está registrado en la plataforma.

Para la gestión de los datos entre la capa de presentación anteriormente descrita y la capa de negocio hemos usado un *Transfer*. Este *Transfer* lo utiliza nuestro *Application Service* para trabajar con los datos de la aplicación, comprobar las reglas de negocio y devolver los resultados obtenidos.

En este proyecto no hemos necesitado programar la capa de integración porque, tal y como hemos comentado antes, utilizamos objetos de negocio implementados como entidades JPA.

A continuación vamos a mostrar los diagramas de clases correspondientes al módulo de usuario de la capa de negocio. Estos diagramas están diseñados siguiendo las características de *UML*, definiendo las relaciones entre las clases. Comenzaremos con el diagrama de clases del *Servicio de Aplicación de usuario*. Figura 30.

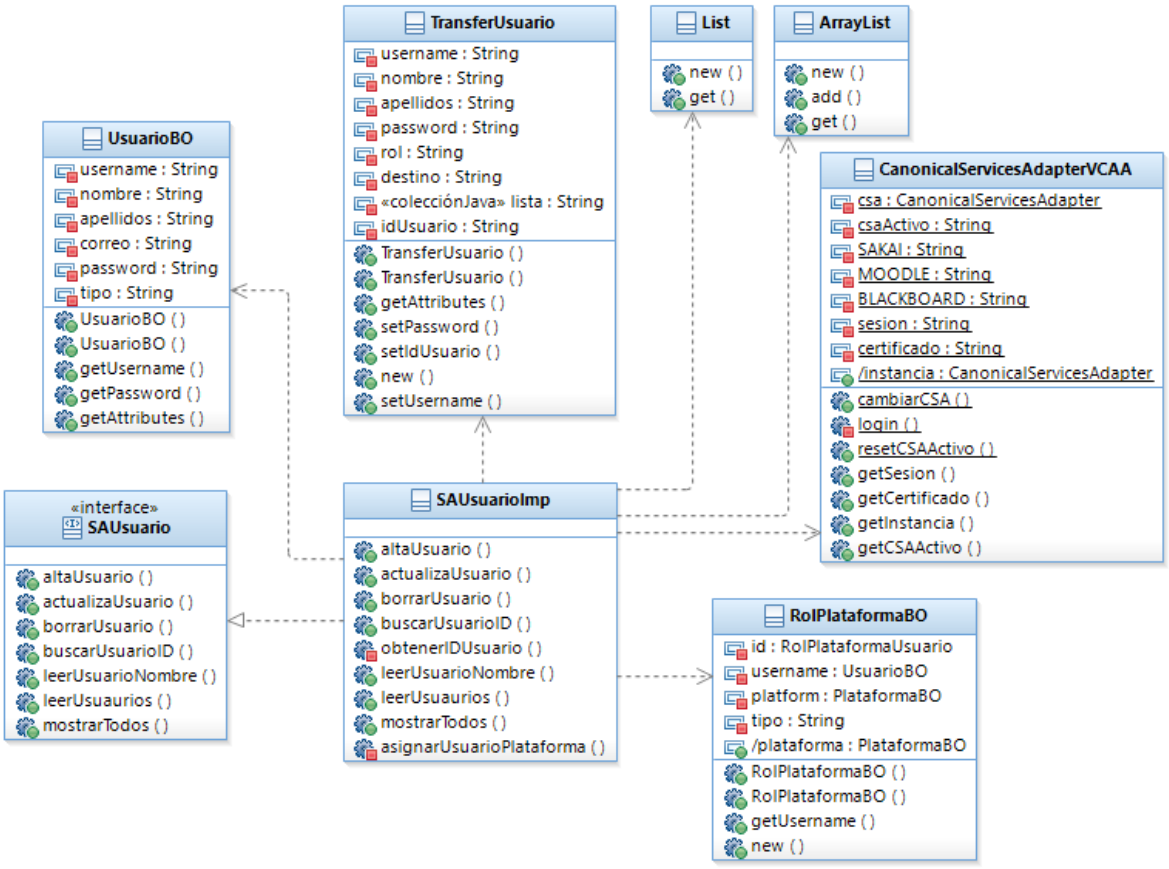

Fig 30. Diagrama de clase de servicio de aplicación de usuario.

Podemos observar que este diagrama describe la compleja estructura de la capa de negocio. También se observa que el *Application Service* es una de las clases más importantes dentro de la actual capa debido a que es el encargado de manejar las reglas de negocio.

Otra parte importante de esta capa es la de las clases *Java* encargadas de las transacciones y de la persistencia de datos. El siguiente diagrama de clases presenta la estructura seguida con estas clases. Figura 31.

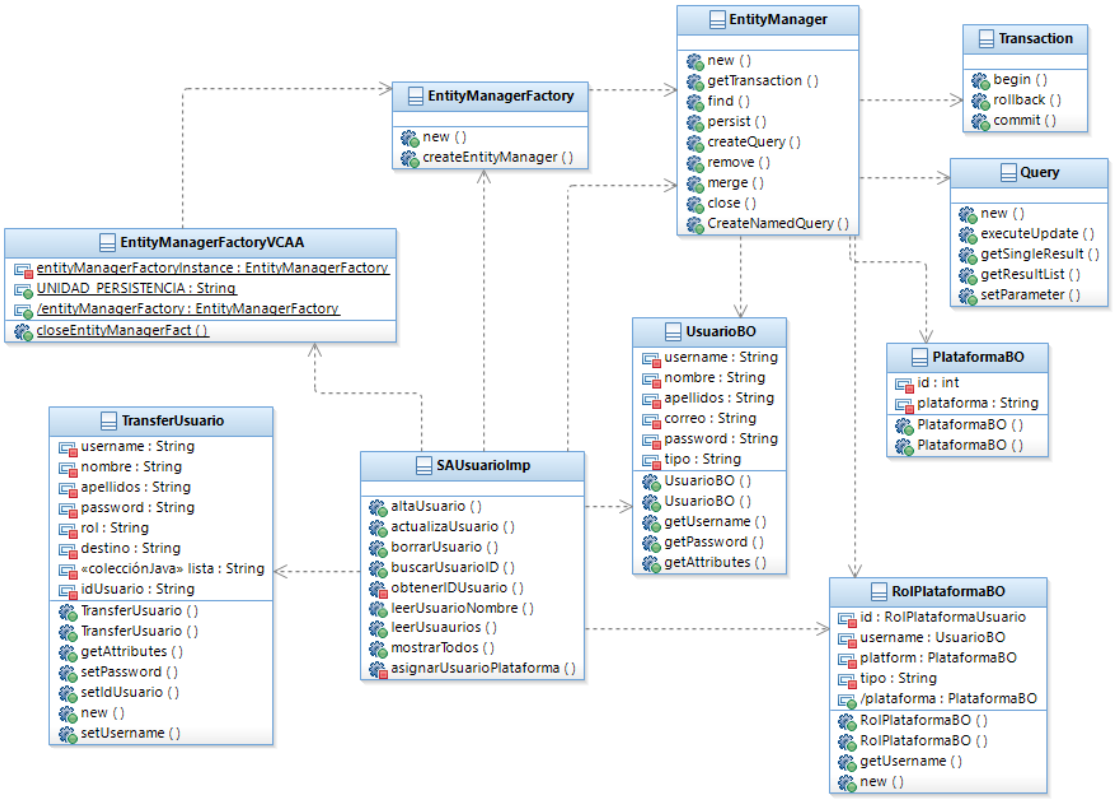

Fig 31. Diagrama de clase de transacciones y persistencia de datos.

También sería idóneo mostrar el diagrama de clases que define la estructura para invocar a los servicios web definidos para las plataformas. Este diagrama incluye contenido desarrollado por Francisco Huertas. Esta estructura usa adaptadores que son *Interfaces* de *Java* para ir definiendo adaptadores para la invocación de los servicios diferenciando así adaptadores para el módulo de usuario y para los demás módulos.

Al ser una estructura compleja, hemos decido crear un adaptador que se imponga por encima de sus adaptadores, de esta manera podemos reutilizar la invocación a los servicios web. Figura 32.

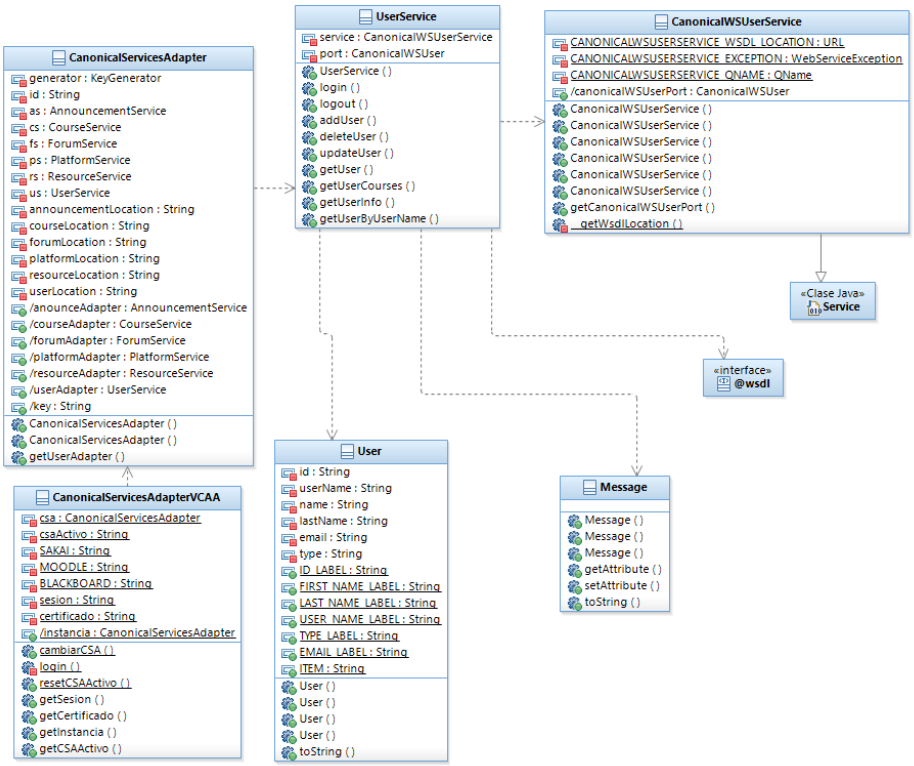

Fig 32. Diagrama de clase de Service Adapter.

Una vez definidos en esta memoria estos diagramas, podemos pasar a definir los diagramas de secuencia. Estos diagramas los vamos a presentar para detallar de una manera técnica y precisa que hacen exactamente los métodos del módulo usuario. Como los anteriores diagramas de clase, estos también siguen el modelado de *UML*. Figuras 33-38.

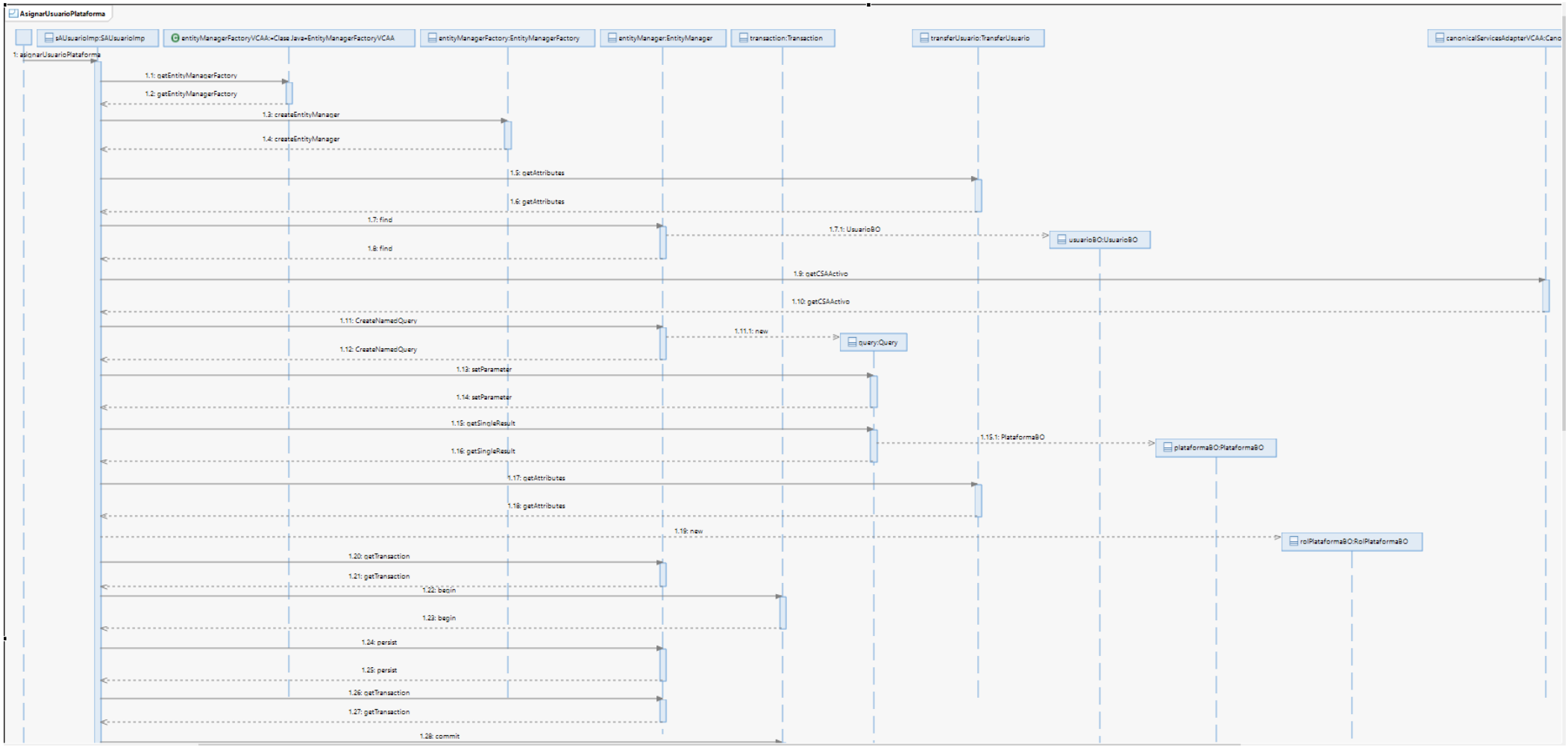

Fig 33. Diagrama de secuencia de Asignar usuario a la plataforma

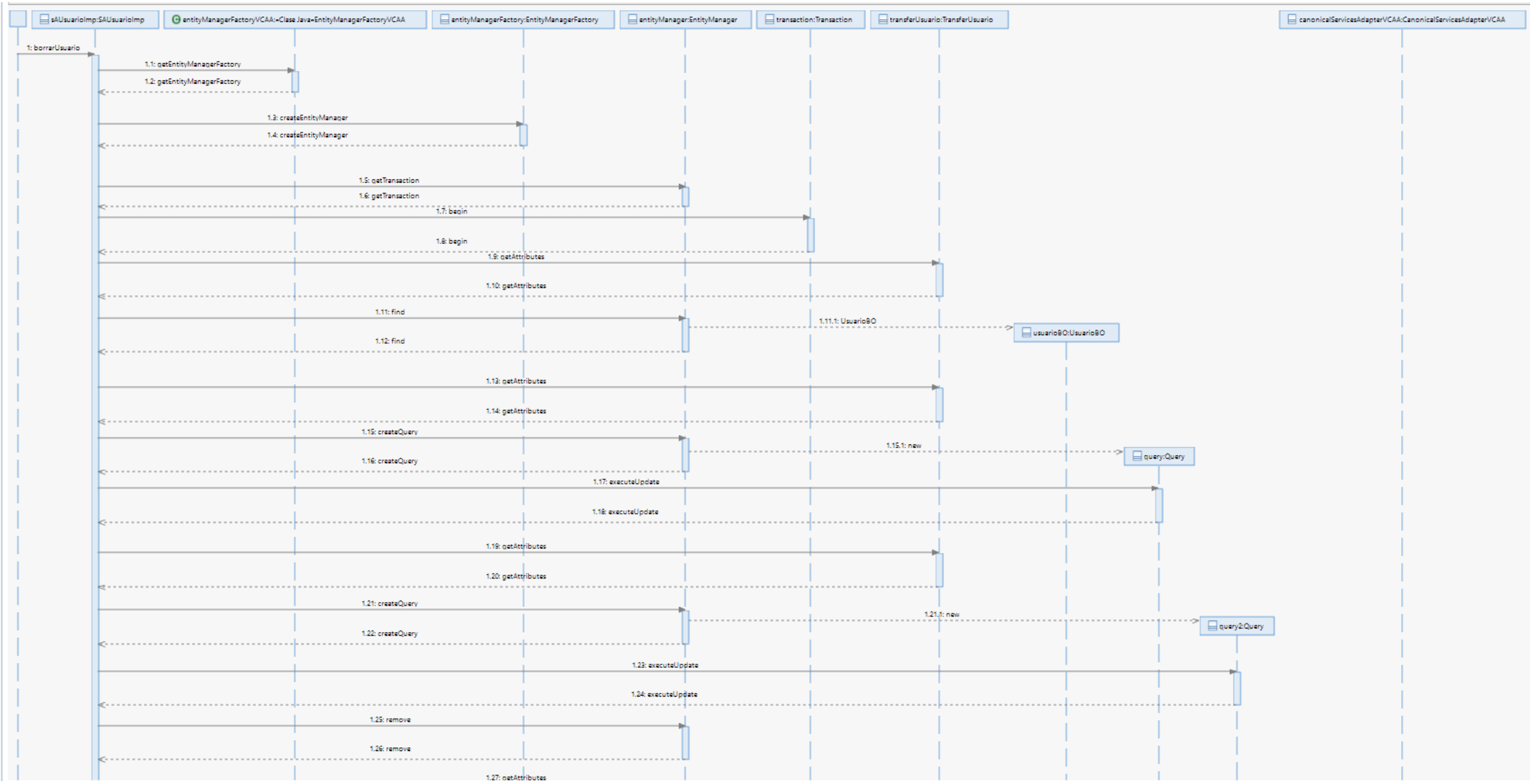

Fig 34. Diagrama de secuencia de Borrar Usuario. Parte 1.

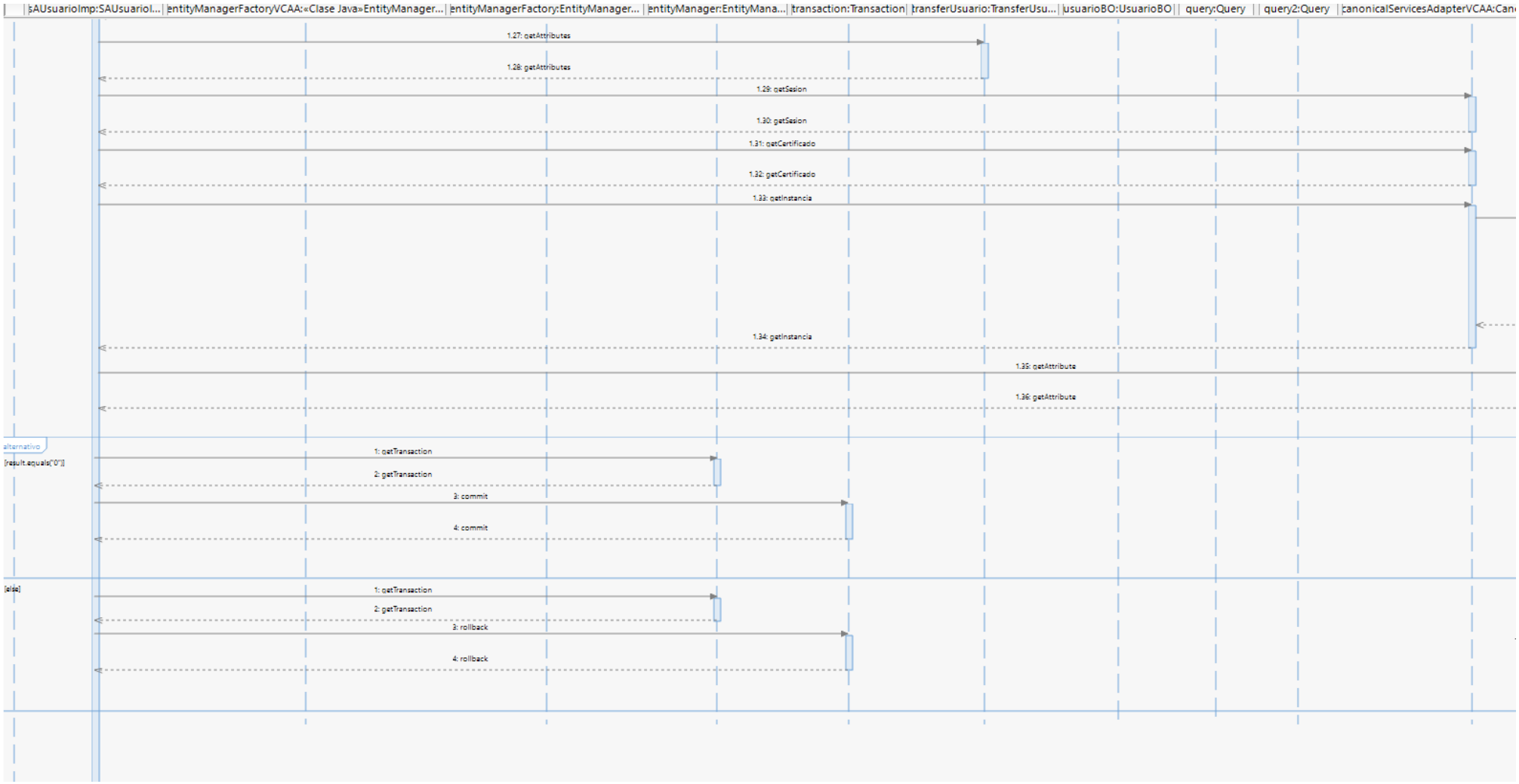

Fig 35. Diagrama de secuencia de Borrar Usuario. Parte 2.

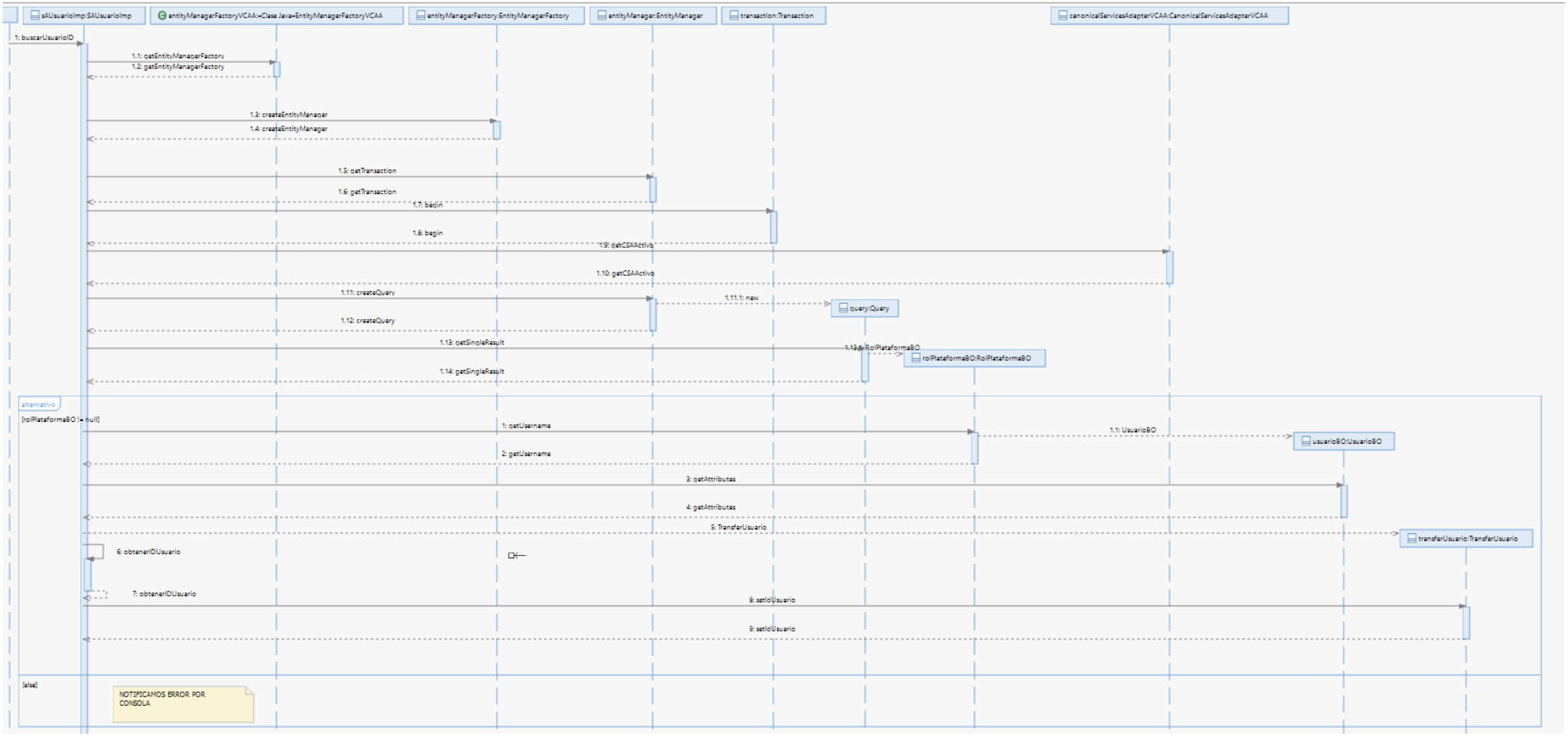

Fig 36. Diagrama de secuencia de Mostrar Usuario.

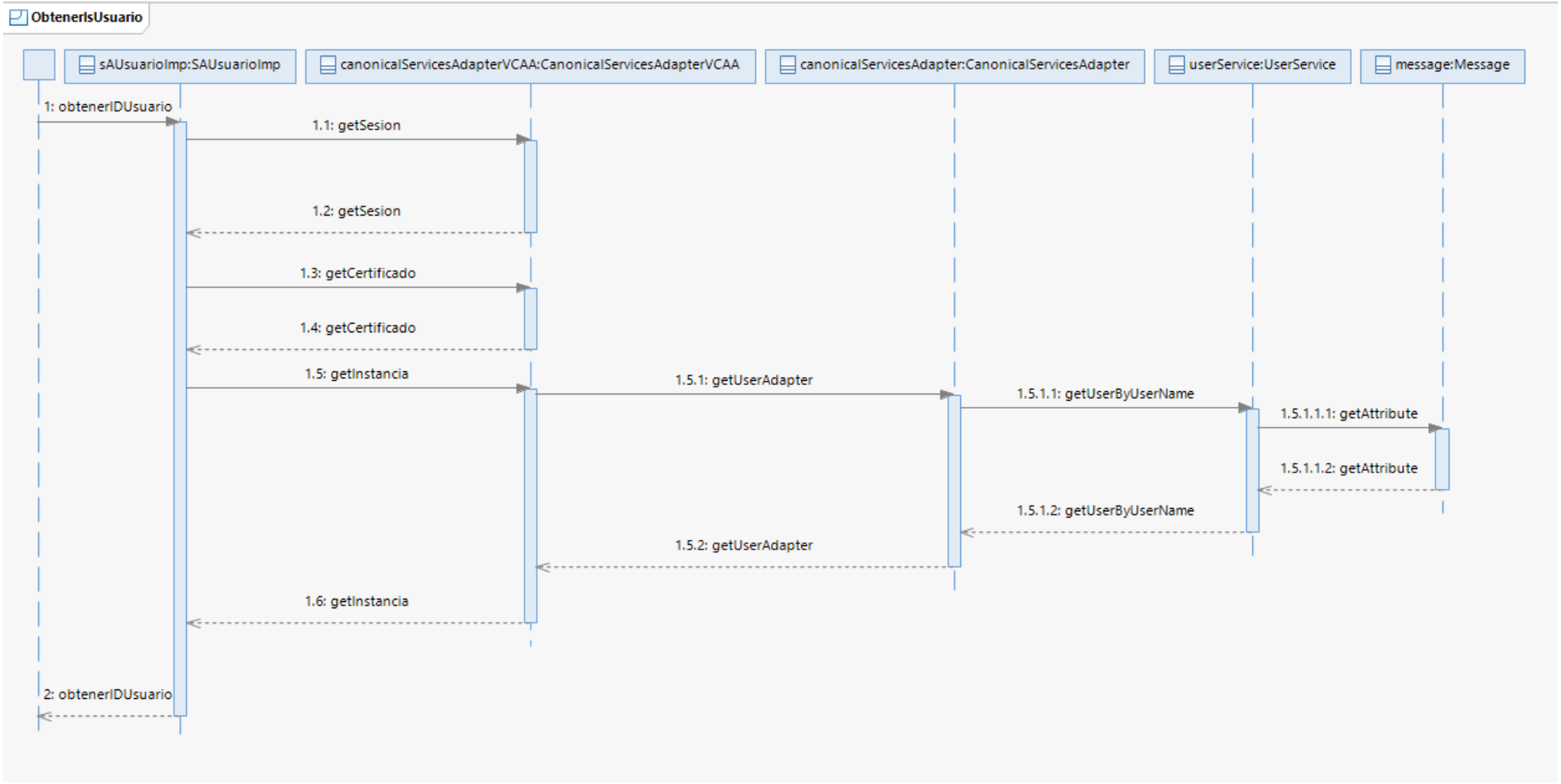

Fig 37. Diagrama de secuencia de Obtener ID Usuario.

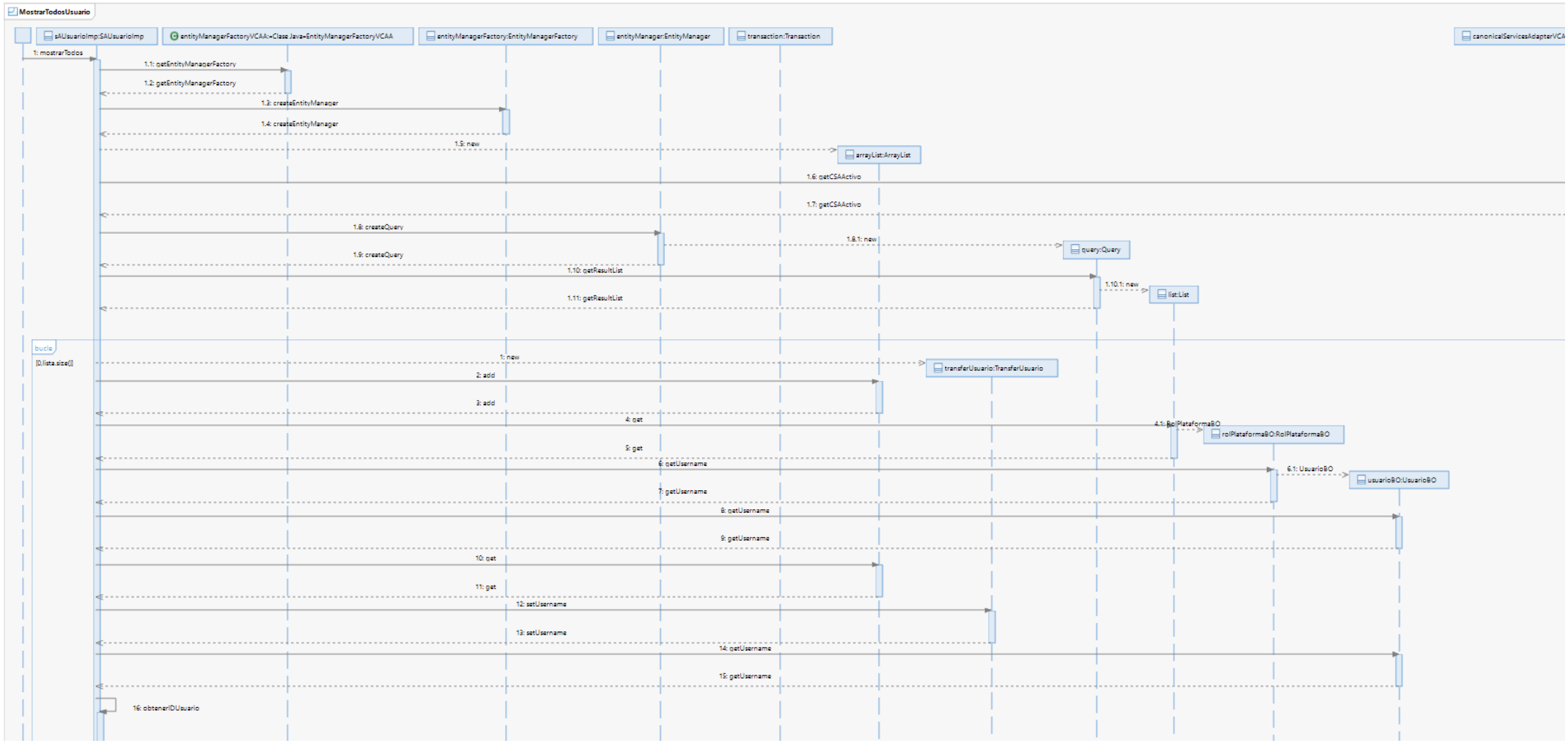

Fig 38. Diagrama de secuencia de Mostrar todos los Usuarios.

**Nota: No se han incluido todos los diagramas de secuencia debido a que su tamaño excedía el tamaño de las páginas y no se visualizaban correctamente.*

3.7 Aportación del equipo

En esta sección pasamos a describir el reparto de la carga que se ha llevado a cabo. Cada miembro del grupo expondrá en que ha estado trabajando a lo largo de este proyecto. Al comenzar este trabajo de fin de grado se decidió una estructura jerárquica horizontal, además de que la carga de trabajo se repartiera entre todos. De esta manera si un miembro necesitaba ayuda con algún problema de una parte concreta del proyecto, todos los miembros del equipo sabrían ayudarle. No obstante no se ha optado por especializaciones pero se ha dejado libre elección sobre qué parte aportar más carga de trabajo en función de las preferencias o habilidades de cada uno. A continuación se presenta a los integrantes del equipo por orden alfabético.

Daniel Martín Barrios:

Al principio del proyecto me documenté de lo que eran los *Java Server Faces* (Hansen, 2007), para aprender lo básico acerca de ellos. En este libro también aparecía un breve capítulo describiendo cómo funcionaban los *Java Beans* junto con *JSF* (Hansen, 2007). Gracias a esta documentación pudimos empezar a visualizar cómo iba a funcionar el proyecto. La primera aportación que hice fue trabajar junto al equipo en un proyecto de prueba menor, en el que se nos pedía la implementación de la navegación web mediante *JSF* y el un simple servicio web para aprender a invocarlos. Esta prueba fue muy dirigida por el tutor.

Una vez completado esto, nos pusimos a realizar un estudio sobre cómo funcionaba el campus virtual que estaba en uso y que debíamos cambiar. Analizamos la funcionalidad que tenía y diseñamos unos diagramas de actividad. Una vez completado esto se nos facilitó una demo para poder probar los servicios web. Para avanzar más rápido nos dividimos la creación de una nueva demo, en la que únicamente se invocaba a los servicios web de las plataformas a desplegar por debajo. Yo asumí la carga de las pruebas en *BlackBoard*.

Para comenzar con el proyecto, usamos mi ordenador portátil como equipo de pruebas, se acordó una reunión con Antonio Navarro para que nos enseñara cómo configurar eclipse para el correcto desarrollo de aplicaciones *JSF*. La configuración quedó reflejada en mi portátil y a partir de aquí

tendríamos que configurar los demás equipos. Este proceso costo mucho, ya que al principio el entorno de mi portátil era inestable, y ningún miembro del equipo sabía con seguridad la razón del por qué.

Tras este periodo una vez con los entornos de desarrollo configurados correctamente empezamos a construir nuestro proyecto, comenzando con un esqueleto basado en el proyecto anterior de pruebas que habíamos realizado. Comencé con la estructura de la parte de la presentación del módulo de usuario, y a partir de esta se fueron implementando las demás. El esqueleto de este módulo en su mayoría lo realicé porque los demás miembros del equipo no disponían del tiempo suficiente en esas fechas y así el tiempo que estuviéramos los tres miembros del equipo trabajando a la vez, sería mucho más productivo. Para la parte de negocio se optó por seguir los patrones de ingeniería multicapa. Aquí he realizado algunas de las operaciones *CRUD* de la entidad Usuario como por ejemplo *alta de usuario*. Una vez acabadas las operaciones *CRUD* de usuario, empezamos con las operaciones de la entidad *Curso*.

Cabe destacar en esta parte que para la invocación de los servicios web, se me ocurrió que podíamos ahorrar carga de trabajo si reutilizábamos parte del código de Francisco Huertas, usando solamente los conversiones de las respuestas y las invocaciones tal y como las tenía Francisco Huertas.

Una vez aclarado esto he sido el encargado de la inclusión de JPA en el proyecto para poder persistir los datos en nuestra base de datos. Una vez incluido este marco, desarrollé como ejemplo dentro del módulo de usuario como sería la persistencia y a partir de aquí se desarrolló para todos los demás módulos.

Días después de esto, el equipo donde se realizaban las demos, que era mi portátil, tuvo problemas técnicos y tuve que formatearlo. Esto me llevó fuera del proyecto durante una semana debido a que al ser el equipo de las demos era necesario para poder demostrar el avance en el proyecto. Durante este periodo se empezó el diseño de la interfaz de usuario de la aplicación en la que no participé a penas dado que el equipo de demos tenía una relevancia superior. Tras varios intentos de arreglar el equipo portátil, opté por instalarme el entorno en mi equipo personal de sobremesa y a partir de aquí ya volví a desarrollar.

Al volver, la funcionalidad del proyecto ya estaba más avanzada, me encargué de la parte correspondiente a la persistencia entre usuarios, plataformas y cursos. Para esto tuvimos que sacar información de nuestros apuntes de asignaturas del grado que nos facilitasen ejemplos y datos sobre cómo hacer las relaciones entre las tablas de la base de datos usando JPA.

Una vez acabado mis compañeros me explicaron cómo funcionaba toda la capa de presentación,

cómo había quedado la interfaz de usuario y cómo usarla. Tras esto también ayudé, aunque poco en comparación con lo que había hecho anteriormente de *Java*. Juan Antonio y yo nos dimos cuenta de que la librería que estábamos usando de *JSF* estaba desaprobada y que había funcionalidad que necesitábamos de la que no disponía. Una vez actualizada, seguimos desarrollando las entidades de Anuncios, Foros y Recursos. De estas tres entidades, en la que más he participado ha sido en Anuncios y en Foros. Los últimos retoques y arreglos de bugs fueron llevados a cabo en su gran medida por Juan Antonio debido a que Javier y yo estábamos de periodos intensos de prácticas en la Universidad.

Una vez acabada la aplicación empezamos con el desarrollo de la memoria. Aquí he aportado también en gran medida porque tengo mayor facilidad para redactar. Todos los miembros hemos aportado en la memoria dado que es un documento extenso y necesita de los conocimientos de todos. Para el desarrollo de la memoria y su correcta ilustración se han diseñado unos diagramas de clase, diagramas de secuencia y diagramas de actividades para facilitar la comprensión. He sido el encargado de realizar los diagramas de actividades. En cuanto a los diagramas de secuencia, los hicimos entre los tres miembros del grupo dado que eran los más complejos. Los diagramas de clase han sido también diseñados entre los tres miembros del grupo.

Javier Mendoza Gómez:

En un principio debido a que no poseíamos suficiente conocimiento en las tecnologías que íbamos a usar en el proyecto, nuestro tutor Antonio Navarro, nos dio una serie de libros para ayudarnos a entender dichas tecnologías, los libros fueron dos; Core JavaServer™ Faces, Third Edition, Video Enhanced Edition- *David Geary; Cay S. Horstmann* y SOA Using Java™ Web Service- *Mark D. Hansen.*

A mí me correspondió la lectura del libro SOA Using Java™ Web Service - *Mark D. Hansen,* la lectura de este libro nos permitió entender el funcionamiento de los servicios SOAP, la diferencia que existía entre ellos y los servicios REST, así como también la publicación de los servicios SOAP.

Debido a que contábamos con los servicios web que habían sido creados en el proyecto anterior de Francisco Huertas y Antonio Navarro no nos fue necesario crearlos, sino simplemente invocar los servicios SOAP, enviando los datos en formato WSDL y recibiéndolos en el mismo formato.

Para que todos aprendiéramos a invocar dichos servicios web nos dividimos el trabajo en invocar las funcionalidades de los servicios web de cada plataforma. Esta división de trabajo nos permitió darnos cuenta de que existían algunos servicios web que no funcionaban como estaba previsto, lo que complicó todo el proyecto.

Antes de realizar empezar a realizar el código del proyecto tuvimos varias reuniones con nuestro tutor para definir bien las funcionalidades que tendría nuestro proyecto.

Participé junto a mis compañeros y tutor en el análisis de entidades que íbamos a guardar en nuestra base de datos y como documento oficial, entregamos una *SRS*, a nuestro tutor.

Dicha SRS contenía las funcionalidades de la aplicación, tanto a nivel de sistema como a nivel de usuario, diagramas de actividades y modelo del dominio del proyecto. Yo, al igual que mis compañeros participamos en todas las fases de este documento.

En un principio nos costó mucho la configuración de JAAS en nuestro proyecto ya que había que configurar muchas variables de entorno para su correcto funcionamiento. La inclusión de JAAS nos permite configurar los accesos a determinados recursos simplemente modificando el archivo web.xml.

Además de ello podemos configurar nuevos perfiles de forma fácil para protegernos de usuarios indeseados. Para el inicio de sesión contra los web services con JAAS, nos basamos en la documentación proporcionada de la web https://jaasbook.wordpress.com.

La configuración de los roles y las variables a configurar para el correcto funcionamiento de JAAS fue mi cometido.

Mi aportación se centró en la funcionalidad de la entidad gestión de recursos. Este gestor nos dio problemas debido a problemas por incompatibilidad de la librería JSF.

Opté por presentar los archivos de manera que se podían crear subcarpetas dentro de la carpeta raíz para poder mejorar, la presentación de los archivos. Me encargué de las operaciones de mostrado de archivos, mostrado de ficheros, descarga de ficheros, borrado de carpetas o archivos, subida de archivos, creación de carpetas, subcarpetas y la navegación entre ellas.

Tantos mis compañeros como yo hemos participado en todos los puntos en la realización de esta memoria.

Juan Antonio de la Vega Gutiérrez:

Al comenzar el proyecto, al igual que mis compañeros, inicié un proceso de aprendizaje leyendo la documentación aportada por nuestro tutor Antonio Navarro. También investigué a través de foros y artículos variados de internet con el objetivo de realizar mis primeras demos para ir cogiendo soltura con JSF.

En la primera demo que realizamos los compañeros y yo para probar la invocación y funcionamiento de los servicios web se me asignaron las pruebas en la plataforma *Moodle*. Debido a un fallo en la hora de los servidores de *Moodle* no se pudo probar la correcta funcionalidad de los servicios web invocados en esta demo inicial.

Las siguientes semanas después de la demo inicial creamos en grupo las operaciones CRUD de la capa de negocio de la entidad Usuario. Estas primeras versiones ya usaban JSF para la entrada y la salida de datos por pantalla, aunque la interfaz era muy pobre, y estaba ampliamente apoyada por los datos mostrados por consola.

Al disponer de más tiempo que mis compañeros empecé a desarrollar los primeros esbozos de una interfaz gráfica más cercana a los marcos actuales. Debido a la entrega de los demás entidades de la aplicación el desarrollo de la interfaz quedó relegado a un segundo plano. Tiempo después, al retomar la interfaz gráfica me di cuenta de que no era suficientemente intuitiva y de que quedaba lejos ser *Material Design.* Para esta nueva vuelta de tuerca a la interfaz opté por colores más apagados obtenidos de https://material.io/guidelines/style/color.html mezclados con grises para no sobrecargar la vista. Los elementos importantes se destacan en un color más claro para llamar la atención y con sombreado para que sobresalgan por encima de los demás elementos. Opté por imitar las opciones que se mostraban en la interfaz antigua pero usando desplegables y menús en vez de pestañas. Inicialmente las opciones del menú lateral estaban ocultas y al dar al botón de menú se mostraban desplazando el contenido de la página a la derecha. Como esto hacía que los elementos importantes perdieran el centro de la página lo cambiamos para que el menú lateral no moviera el contenido. Nos dimos cuenta de que si el usuario debe realizar varias acciones en la aplicación es molesto tener que pulsar continuamente el botón de menú, por lo tanto cambié el menú lateral para que apareciera desplegado por defecto y que se pudiera ocultar en caso de ser necesario. Además de eliminar el desplazamiento del contenido para que este estuviera siempre centrado en pantalla.

Para las acciones a realizar (como borrar, editar y visitar) así como para el apoyo de los elementos del menú lateral propuse una serie de iconos obtenidos de la página http://getbootstrap.com/components/ de iconos de *Bootstrap*. Estos iconos, al ser intuitivos, permiten seleccionar las opciones de los menús prácticamente sin leer el texto.

Como la selección de la plataforma es algo fundamental en nuestra aplicación, creé una página intermedia entre el login (con resultado exitoso) y la página de Home, con el objetivo de que el usuario se viera obligado a elegir una plataforma en la que trabajar, y evita así esa página prácticamente vacía a la que llegaba un usuario que no había elegido plataforma al loguearse en la interfaz antigua. Acompañé los botones que hacen login sobre una u otra plataforma con una imagen representativa de esta y con la misma funcionalidad para agilizar el flujo.

Las últimas semanas se dedicaron a hacer que las funcionalidades de la aplicación, que mayoritariamente se habían probado para la plataforma Sakai, funcionaran para el resto de plataformas. Como cada plataforma gestiona los identificadores de una manera, hacer que la aplicación fuera totalmente funcional en todas las plataformas fue algo más laborioso de lo esperado. También me dediqué a resolver bugs de funcionalidad y diseño durante este tiempo.

Para la realización de la memoria hemos participado todos los miembros del equipo para asumir la gran carga de trabajo que esto supone.

Conclusiones

Tras haber realizado este proyecto hemos llegado a la conclusión de que hemos tenido que aprender muchas tecnologías nuevas que ningún miembro del grupo conocía, como *JSF* o *JAAS*. Este proceso de aprendizaje siempre conlleva un gran esfuerzo porque son tecnologías que no hemos usado a lo largo de la carrera.

Como todo proceso de reingeniería, este proyecto ha resultado más complejo de lo que en un principio se estimó. Esto ha sido provocado porque la documentación de la aplicación no favorecía su mantenibilidad. En un principio se optó por una metodología tradicional dirigida por modelos, pero debido a los múltiples problemas que se han dado a lo largo del proyecto se ha optado por una metodología ágil.

A pesar de esto, se ha reutilizado la invocación existente de los servicios web, que dentro de lo que cabía era una parte que seguía una estructura lógica. Esto ha facilitado el proyecto y la carga que ha supuesto al equipo.

La aplicación ahora cuenta con *servicios de aplicación* (SA) bien definidosque estructuran las reglas de negocio de la aplicación. Esto organiza mejor toda la funcionalidad de la aplicación y añade la posibilidad de publicar como servicios web los diferentes métodos de cada servicio de aplicación. Además la aplicación cuenta con una nueva interfaz de usuario más actualizada en el marco actual de interfaces.

Tras la realización de este proyecto, el equipo de desarrollo ha adquirido más madurez para afrontar problemas distintos a los planteados a lo largo del grado. El equipo ha aprendido a lidiar con problemas de estabilidad en el entorno de desarrollo, también a aprender tecnologías que ningún miembro del grupo comprendía. Además se ha mejorado la capacidad de cada uno de los miembros para trabajar en equipo.

Como conclusión final asignaturas como *Modelado de Software* e *Ingeniería del Software* consideramos que son muy importantes debido a que te enseñan cómo construir una arquitectura multicapa. También consideramos que en la realidad, una arquitectura multicapa es esencial para la estructuras *software* debido a que ésta facilita posibles mejoras y extensiones que queramos hacer en nuestra aplicación, sin contar con que a la hora del entendimiento del código es mucho más sencillo para los programadores o ingenieros.

Agradecemos al tutor Antonio Navarro Martín toda la ayuda que nos ha prestado a lo largo del desarrollo del proyecto.

Conclusions

After realizing this project we have come to the conclusion that we have had to learn many new technologies that no member of the group knew, like *JSF* or *JAAS*. This learning process always involves a great effort because they are technologies that we have not used throughout the grade.

As any reengineering process, there has been more effort than was initially estimated. This has been caused by lack of a good documentation. Initially we opted for a traditional model-driven methodology, but due to the multiple problems that have occurred in the life of this project, we have opted for an agile methodology.

In spite of this, the existing invocation of web services has been reused, which within it was a part that followed a logical structure. This has facilitated the project and the burden on the team.

The application has now well-defined *application services* (SA) that structure the business rules of the application. These *application services* organize all the functionality of the application and add the possibility of publishing as web services the different methods of each application service. In addition the application has a new user interface more updated in the current framework of interfaces.

After finishing this project, the development team has acquired more maturity to resolve different problems from those raised throughout the grade. The team has learned to deal with stability issues in the development environment, as well as to learn technologies that no member of the group understood. In addition, the ability of each member to work as a team has been improved.

As a final conclusion, we consider that subjects such as Software Modeling and Software Engineering are very important because they teach you how to build a multi-layer architecture application. We also consider that in reality, a multilayer architecture is essential for software applications because it facilitates possible improvements and extensions that we want to make in our application, making understanding code much easier for programmers or engineers.

Índice de Figuras:

Referencias

- Alur, D., Crupi, D., Malks, J. (2003). *Core J2EE Patterns: Best Practices and Design Strategies (2nd Edition)*. Prentice Hall, 2003.
- Coward, D. (2014). *Java EE 7: The Big Picture*. McGraw-Hill Education, 2014.
- Erich Gamma, Richard Helm, Ralph Johnson, John M. Vlissides (1994). *Design Patterns: Elements*
    - *of Reusable Object-Oriented Software.*
- Geary, D., Horstmann, C.S. (2010). *Core JavaServer Faces. 3rd Edition*. Prentice-Hall, 2010.
- Hansen, M.D. (2007). *SOA Using Java Web Services*. Prentice Hall, 2007.
- Huertas, F., Navarro, A. (2015). SOA support to virtual campus advanced architectures: The VCAA canonical interfaces. *Computer Standards & Interfaces* 40: 1-14 (2015).
- IEEE Computer Society. Software Engineering Technology Committee, Institute of Electrical and Electronics Engineers (1998). *IEEE Recommended Practice for Software Requirements Specifications*.
- Navarro, A., Cristóbal, J., Fernández-Chamizo, C., Fernández-Valmayor, A. Architecture of a multiplatform virtual campus. *Software Practice and Experience* 42(10): 1229-1246 (2012).
- Newcomer, E. (2002). *Understanding Web Services: XML, WSDL, SOAP and UDDI*. Addison-Wesley, 2002.
- Oracle (2017). *Java Authentication and Authorization Service (JAAS) Reference Guide*. https://docs.oracle.com/javase/8/docs/technotes/guides/security/jaas/JAASRefGuide.html
- Pressman, R.S., Maxim, B (2014). *Software Engineering: A Practitioner's Approach (Irwin Computer Science) 8th Edition*. McGraw-Hill Education, 2014.