APLICACIÓN WEB DE APOYO A LA INVESTIGACIÓN Y DOCENCIA DE LA INVESTIGACIÓN

UNIVERSIDAD
COMPLUTENSE
MADRID

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA DE COMPUTADORES

CURSO 2017-2018

CONVOCATORIA DE FEBRERO

Autor: Carlos Villarroel González

Director: Juan Pavón Mestras

Departamento de Ingeniería del Software e Inteligencia Artificial

Facultad de Informática

Universidad de Complutense de Madrid

Agradezco a Juan Pavón el haber aceptado y mejorado enormemente Medievalia.

Y a mi familia por su preocupación, especialmente a Óscar Villarroel González, por ser un buen "cliente final", pidiendo mucho cuando debía, y poco cuando podía. Y a Magdalena A. Marañón Palma, por su ayuda en el proyecto y sobre todo por ayudar en casa, con Víctor, aguantando el estrés.

A Víctor, por estar. Para cuando puedas leerlo, que haya servido de algo.

ÍNDICE DE CONTENIDO

RESUMEN

Medievalia es una aplicación web que, más allá de ser una base configurable de conocimiento para la investigación, es una herramienta para que alumnos de grado o máster de diferentes áreas aprendan las técnicas de investigación que utilizan sus profesores.

Ofrece un entorno donde, además de poder compartir conocimiento entre todos los usuarios, los profesores pueden tutorizar, validar, corregir y comentar las aportaciones realizadas por los alumnos. Alumnos y profesores contribuyen a aumentar la información que es almacenada en la base de datos.

El entorno web fue desarrollado con un diseño *responsive* y cuidando la accesibilidad, usando las herramientas más populares para este propósito, siendo accesible desde cualquier lugar y desde cualquier navegador web.

El hecho de que sea personalizable implica que, a diferencia de una base de datos común y una interfaz de usuario para manejarla, los diferentes tipos datos que se almacenarán y sus relaciones pueden ser definidas para diferentes entornos de estudio como la historia, sociología, etc.

Finalmente, esta aplicación web fue probada con el esquema por defecto por alumnos y profesores del grado de Historia, en la facultad de Geografía e Historia. El resultado fue satisfactorio.

Palabras clave: **Investigación**, **Base de Datos**, **Sistema de Gestión de Aprendizaje**, **Aplicación Web**

SUMMARY

Medievalia is a web application with a customizable base of knowledge for research and a tool for undergraduate or masters students from different disciplines to learn the research techniques used by their teachers.

It offers an environment to share knowledge among all users, and teachers can supervise, validate, correct and comment on the contributions made by students. Both students and teachers contribute to increase the amount of data stored in the database.

The web environment was developed with a responsive design and taking care of the accessibility, using the most popular tools for this purpose, being accessible from everywhere and from any web browser.

Being a customizable application implies that, unlike a common database and a user interface to manage it, the different types of data that will be stored and their relationships can be defined for different researching areas such as history, sociology or others.

Finally, this webapp was tested with the default schema by students and teachers of the degree in History, in the Faculty of Geography and History. The result was succesful.

Keywords: ***Research***, ***Database***, ***Learning Management System***, ***Webapp***

1 INTRODUCCIÓN

1.1 ANTECEDENTES

Más de una década antes que el presente proyecto si quiera se plantease, el doctor Óscar Villarroel González, profesor en la facultad de Geografía e Historia, desarrolló una base de datos en MS Access para su uso personal en la investigación histórica. Pese a contar con ciertos conocimientos informáticos y ser útil finalmente, esta base de datos MS Access con interfaz gráfica tuvo limitaciones e incomodidades que un alumno de cualquier grado de informática tras cursar (y aprobar) bases de datos podría resolver.

De un simple esquema entidad-relación comenzaron a surgir posibilidades prometedoras en caso de desarrollarse como una aplicación web. De esta forma se llegó a una versión anterior de Medievalia, realizada en PHP pero que no llegó a terminarse. Ya en 2014, cursándose la adaptación al Grado de Ingeniería de Computadores, se propuso esta primera versión como Trabajo Fin de Grado. Fue en ese momento donde de una interfaz web para una base de datos se convirtió casi a la aplicación web que ahora es, yendo mucho más allá de ser una base de datos, incluyendo la parte docente, la relación alumno-profesor, pudiéndose calificar en la práctica como LMS (*Learning Management System*).

Esta ampliación abrió nuevos horizontes para la aplicación, como el hecho de ser presentada como un Proyecto de Innovación Docente. Esto será dilucidado durante el siguiente curso 2018-2019.

La situación laboral como docente impidió que se terminase el trabajo final de grado en ese mismo curso, pese a aprobar el resto de asignaturas. Añadido a esto, la situación legal que equiparaba las ingenierías técnicas con los grados, restó urgencia al proyecto. Sin embargo, esa misma situación laboral permitió aprovechar cierto tiempo de desarrollo, sobre todo durante vacaciones de verano, donde, como se verá, un alto porcentaje del trabajo fue realizado.

De esta forma se llega a 2017, donde la aplicación llegó a un punto de desarrollo cercano a su finalización, motivando la matriculación del proyecto, un último impulso (duro, compaginado como anteriormente a obligaciones laborales y familiares) y a nuevas modificaciones del proyecto original que hacen de él una herramienta mucho más atractiva, como el hecho de ser flexible a las necesidades de diferentes áreas de investigación.

1.2 Bases tecnológicas

Tratándose de una aplicación web se ha hecho uso de un alto número de tecnologías diferentes para cada uno de los aspectos diferentes de la aplicación.

- Servidor web Apache Tomcat 7.0.
- Durante el desarrollo inicial de la aplicación desde octubre de 2014 el servidor se encontraba en un Ubuntu Desktop 14.04 en un portátil Dell XPS Developer Edition, procesador Intel Core i5 5200U a 2.2 GHz con 2 núcleos, 4 subprocesos y 8Gb de memoria principal.
- El servidor de producción, finalmente en noviembre de 2017, fue un openSUSE 13.2 (Harlequin), bajo un equipo con un Intel Xeon E5504 a 2 GHz, 4 núcleos, 4 subprocesos y 4Gb de memoria principal (2Gb libres con el equipo recién iniciado).
- La propia aplicación se ha desarrollado con el lenguaje Java, usando la JDK 1.8.0_151. Las vistas están desarrolladas con JSP y diferentes librerías de jstl.
- Spring Framework como entorno de trabajo, especialmente para el desarrollo del modelo-vista-controlador.
- Mysql para la persistencia de los datos. Tanto en desarrollo como en producción el sistema gestor de la base de datos se encontraba en el mismo computador aunque, como es habitual, puede encontrarse separado.
- El navegador del usuario hace uso de Jquery, AJAX y JavaScript, y de forma especialmente intensiva en algunas pantallas.

1.3 Metodología

Gracias a la experiencia previa en desarrollo web se ha usado una metodología de desarrollo ágil. Con esta metodología el desarrollo fue iterativo e incremental. Cada iteración supuso una planificación propia, análisis de requisitos, diseño, codificación, documentación y pruebas.

En la práctica no todas las iteraciones siguieron todos los pasos, dado que muchas funcionalidades en una aplicación web son similares, con requisitos casi idénticos, siguiendo

un mismo diseño y una serie de pruebas equivalentes.

1.3.1 Desarrollo de la primera iteración

Inicialmente se desarrolló un núcleo con funcionalidad mínima, ya haciendo uso de modelo-vista-controlador, con un sencillo mensaje “hello world”. Posteriormente a este paso se añadieron funcionalidades mínimas o herramientas esenciales:

- Diseño y acceso a base de datos.
- Boceto de la interfaz final con cabecera, Bootstrap y librerías jstl.
- Configuración de debug en eclipse.
- Ventana de login con inicio de sesión provisional y cierre del mismo.

Una vez llegado a este punto se dio por terminado el núcleo desde el cual se añadirían nuevas funcionalidades con cada iteración.

1.3.2 Desarrollo de iteraciones posteriores

El resto de iteraciones siguió un esquema similar en la mayor parte de las funcionalidades añadidas. Cada iteración por lo general resultó en la codificación de un nuevo controlador, o en algunos casos dos si la funcionalidad requería varios pasos. Y solo en contadas ocasiones requería pasos diferentes a los siguientes:

- Análisis: se identificaba qué actores podrían realizar la acción, con qué precondiciones (habitualmente datos en sesión).
- Diseño: evaluación sobre si la funcionalidad tiene efecto sobre el modelo de datos y la información que es necesario transmitir.
- Codificación: realización de cambios en el modelo de datos, y comprobaciones de autorización y autenticación. Comprobación de precondiciones (datos de sesión) y finalmente desarrollo de la propia funcionalidad y su visualización.
- Pruebas: las pruebas de cada iteración se realizaron de dos formas diferentes. Inicialmente con la propia interfaz web, con los datos que esta ofrece, y, posteriormente, con la consola JavaScript, introduciendo datos en las peticiones que podrían ser inconsistentes y que por lo tanto no son accesibles a través de la interfaz.

Generalmente la planificación del trabajo se ajustaba a los requisitos laborales de la forma más flexible, mientras que la documentación en el caso de la mayor parte de las iteraciones no se realizó hasta terminado el proyecto.

1.4 Gestión de riesgos

Se han identificado riesgos durante la instalación y configuración de la aplicación así como del uso de la aplicación completamente configurada y se definen en la siguiente tabla.

Identificación	Descripción	Probabilidad de riesgo	Consecuencias
1	La máquina virtual de Java instalada en el servidor es inferior a 1.5	1	5
2	El servidor no tiene versión compatible para instalar Tomcat o MySql	1	5
3	El servidor se vuelve inaccesible o se ha apagado.	2	4
4	El servidor deja de dar servicio correctamente por alta demanda.	3	2
5	El navegador que usa el usuario pierde la conexión.	3	3
6	El navegador que usa el usuario es incompatible con la versión Bootstrap o JQuery.	1	3

Tabla 1: Tabla de identificación de riesgos

Leyenda:

- Probabilidad:
  - 1: Muy improbable.
  - 2: Poco probable.
  - 3: Moderadamente probable.
  - 4: Probable.
  - 5 Muy probable.
- Consecuencias:

- 1: No supone ningún problema.
- 2: Supone un pequeño problema, ralentización o empeoramiento de experiencia de usuario.
- 3: Supone un problema moderado. No es accesible el servicio pero no corre riesgo la información guardada.
- 4: Supone un gran problema. El servicio no es accesible y algunos datos se han podido perder.
- 5: Supone un problema crítico. El servicio no solo es inaccesible, sino que es inviable.

Con los riesgos identificados se expone a continuación el plan de contingencia para cada uno.

Identificador	Plan de contingencia
1	Instalar la última versión de Java disponible.
2	Instalar otro sistema operativo compatible o actualizarlo, dada la larga lista de sistemas compatibles y desde versiones muy antiguas.
3	Contactar con el administrador del servidor para que compruebe el estado del servidor y tome las medidas adecuadas.
4	El servidor dedicado tiene unas bajas prestaciones o su conexión es lenta para la demanda de usuarios, por ello puede inicialmente dosificar el acceso mediante la división en grupos de alumnos más pequeños como medida temporal. Se requeriría un estudio para explorar la posibilidad de cambiar de servidor a uno más potente o incluso externalizarlo.
5	Este riesgo puede ser común si el usuario utiliza la aplicación con el móvil o una tablet durante trayectos, por lo que es habitual en este caso y resulta un problema temporal. Si se da cuando el usuario está en un equipo fijo será necesario revisar su conexión a internet, si es un corte puntual, o si se repite por lo que en este caso habría que contactar con el administrador de la red.
6	Inicialmente a los futuros usuarios de la aplicación se les recomendará el uso de móviles, tablets y equipos que no sean muy antiguos, dado que cualquier sistema de los últimos 6 u 8 años es compatible. Las versiones incompatibles de los navegadores son extremadamente antiguas, por lo que se requerirá la instalación de las últimas versiones o se recomendará el uso de otros equipos más modernos.

Tabla 2: Tabla de planes de contingencia

1.5 PALABRAS CLAVE

Para comprender correctamente la organización y composición de la aplicación se definirán una serie de conceptos mínimos de la aplicación.

1.5.1 Usuario

Un usuario es la entidad con la que un actor que utilice la aplicación se va a identificar. Todas las acciones menos la propia autenticación requerirán esta relación entre usuario y actor.

El actor cuenta con un nombre de usuario, un nombre completo (que será el de la persona que usará la aplicación), un identificador numérico, y un rol.

1.5.2 Rol

Hay una serie de roles predefinidos por defecto los cuales son asignados a los usuarios. Un usuario únicamente podrá tener un rol en cada momento, pero este podrá ser cambiado.

Los roles son usados para definir la autorización de cada usuario. Existe una lista de tuplas rol-acción que definen las posibles acciones que un rol puede realizar. Así, un usuario con un rol tendrá autorización para las acciones que tenga el rol.

Inicialmente están definidos 4 roles: administrador, profesor, alumno e inactivo. Pero modificando la lista de roles-acciones pueden definirse nuevos roles, así como cambiar los permisos de cada rol.

1.5.3 Grupo

Un grupo es una entidad que, en principio, está pensada para ser reflejo de una clase con profesor y alumnos. El grupo tiene un nombre y un director, que es el profesor que ha creado el grupo.

1.5.4 Matriculaciones

Una matriculación es añadir a un usuario a un grupo. Hay dos formas diferentes de matriculación: como alumno y como profesor. Esto no está sujeto a los roles aunque se denominen igual, solo hay estas dos formas de matriculación, pero sí está definido por rol qué tipo de matriculación puede tener un usuario con un rol concreto. Una vez un usuario pertenece a un grupo tendrá acceso a objetos del grupo que aún no han sido validados.

1.5.5 Objetos e instancias de objeto

Los objetos de la aplicación, o también llamados tipos de objeto, están definidos en la base

de datos y por lo tanto son configurables, se pueden añadir nuevos y eliminarlos. Definen cada tipo de entidad con el que los usuarios van a trabajar.

Por defecto están los que se aplican a la investigación en historia: Cargo, Estudio, Lugar, Personaje, Tema, Subtema, Autor, Dato, Documentación y Archivo.

Cada usuario podrá crear instancias de cada objeto, que serán los elementos con los que trabajarán mediante las diferentes acciones que se pueden realizar sobre ellos.

Una instancia de objeto tendrá además nombre, creador, grupo e información de validación.

1.5.6 Atributos sencillos e instancias de atributos sencillos

Al igual que los objetos, un atributo sencillo es una definición y también es configurable. En este caso define qué información guarda un objeto. Un objeto tendrá una lista de atributos sencillos que serán datos únicos que tendrá cada instancia de atributo sencillo. Por ejemplo, el objeto del modelo por defecto “Personaje” tiene como atributos sencillos “Fecha de nacimiento”, “Fecha de fallecimiento”, “Otros”, “Lugar de nacimiento” y “Lugar de fallecimiento”.

Hay seis tipos de atributos sencillos implementados en esta versión:

- Número entero
- Número real
- Fecha especial: admite días o días y meses nulos, además tiene un alcance en fechas antiguas que otros formatos no admiten (se describirá en el epígrafe 2.4.8)
- Texto: un texto corto de hasta 255 caracteres.
- Texto largo: un texto de hasta 65535 caracteres.
- Objeto: hará referencia a otro tipo de objeto.

Una instancia de atributo sencillo será un valor único que tendrá una instancia de objeto del tipo que esté definido. Siguiendo el ejemplo anterior, una instancia de Personaje como “Juan” tendrá una instancia de fecha de nacimiento puede ser “5/1423” y una instancia de lugar de fallecimiento “Oviedo” que ha de ser una instancia de un tipo de objeto concreto obligatoriamente.

En la definición del tipo de atributo sencillo se incluye el tipo de atributo y si este es objeto, define también a qué tipo de objeto hace referencia.

1.5.7 Atributos complejos e instancias de atributos complejos

Un atributo complejo funciona de forma similar a un atributo sencillo, con la salvedad de que el tipo de atributo es siempre de un tipo de objeto y, además, es una lista de ellos. Cada atributo complejo tiene un nombre, un tipo padre y un tipo hijo, además de información de relación.

Por ejemplo, el objeto "Personaje" tiene como atributos complejos "Estudios" y "Cargos", que hacen referencia a los objetos con ese nombre. Una instancia de objeto personaje, cuyo nombre sea "Juan II" puede tener en el atributo complejo "Cargos", y dos instancias de atributo complejo "Cargos" que sean "Rey de Castilla" y "Rey de León".

A continuación se ilustra un ejemplo de objeto con cinco atributos sencillos y dos complejos. A su lado, un ejemplo de instancia de objeto con instancias de atributos sencillos y complejos.

Objeto: Personaje			
Atributos Sencillos		Atributos Complejos	
F.Nac	Fecha especial	Cargos	Objeto Cargo
F.Fac	Fecha especial	Estudios	Objeto Estudio
L.Nac	Objeto Lugar		
L.Fac	Objeto Lugar		
Otros	Texto de usuario		

Instancia de objeto: Juan II			
F.Nac	6/3/1405	Cargos	Estudios
F.Fac	22/7/1454	Rey de Castilla	Bachiller
L.Nac	Toro	Rey de Aragón	Doctor en derecho canónico
L.Fac	Valladolid	Infante	
Otros	Texto de usuario		

Tabla 3: Ejemplo de objeto e instancia de objeto

Hay que añadir que los atributos complejos pueden tener objetos, fechas de inicio y fin y páginas relacionadas. En el ejemplo, las fechas de "Rey de Castilla" serían "1406" y "22/7/1454", con un objeto tipo Documentación "Simancas 1" y una página "34-46".

Se ha de notar que el nombre del atributo complejo no tiene por qué ser homónimo con el nombre del objeto referenciado y un ejemplo es que el objeto que viene por defecto llamado "Dato" tiene una lista de personajes y el nombre del atributo complejo es "Implicados".

1.5.8 Relación

Como se dijo antes, un atributo complejo puede guardar información de la relación. Con

relación se hace referencia a la acción de añadir un objeto como un elemento de la lista de instancias de atributo complejo.

Al añadirse una instancia de atributo complejo, según se configure la aplicación, se podrá añadir cierta información a esta relación. La información varía para cada tipo de atributo complejo, pero puede ser la siguiente como máximo:

- Instancia de relación. Cuando se añade una instancia de atributo complejo esta puede llevar una tercera instancia asociada de un tipo concreto de objeto.
- Fecha de inicio de relación. Al igual que el caso anterior, se puede añadir un dato, en este caso una fecha en la cual se considera que se inicia la relación.
- Fecha de fin de relación. En este caso indica la fecha de finalización.
- Página. Es un valor textual.

Todos estos datos son opcionales y configurables. En el caso de la aplicación en historia y tomando el ejemplo anterior de “Juan II”, la instancia de atributo complejo que indica que Juan II es “Rey de Castilla” llevará como relación una instancia de objeto Documentación, con Página: “12-14”, fecha de inicio “1406” y fecha de finalización “22-7-1454”. Esta configuración tiene el objetivo de indicar la documentación de la que se obtiene la información, así como añadir un rango temporal durante el que se aplica la relación.

1.5.9 Validación

Las instancias de objeto y de atributo complejo pueden ser validadas. Este concepto se refiere al estado de estos elementos, que lo harán visible para todos los usuarios y grupos de la aplicación.

Las instancias que sean creadas por un profesor siempre estarán validadas, mientras que las que crea un alumno siempre estarán no validadas.

1.6 Herramientas de ayuda al desarrollo

1.6.1 Eclipse

Toda la codificación en Java con todas sus pruebas y depuración se desarrolló bajo Eclipse, al cual se le añadieron las herramientas necesarias como maven, spring, git, etc.

Como se ha indicado anteriormente, también fue configurado para la depuración en el servidor de desarrollo. No fue posible hacerlo a su vez en el servidor de producción, dado que el administrador de este servidor no dio los permisos necesarios para hacer todas las operaciones que sí pudieron hacerse en el servidor de desarrollo.

1.6.2 Control de versiones

Durante todo el desarrollo, desde el 3 de enero de 2015, se creó una cuenta en GitHub y esta se añadió a Eclipse, desde donde se realizaban todas las actualizaciones. Por lo general lo habitual fueron todo acciones “commit” (un total de 135) y “push”, aunque en unas pocas ocasiones se tuvo que volver a versiones anteriores, sobre todo al principio en 2015.

A fecha de enero de 2018 el proyecto es accesible en: https://github.com/cvillag/medievalia

1.6.3 Spring y Maven

Spring fue el *framework* escogido para el desarrollo de la aplicación web debido a la experiencia previa con él. De las herramientas que ofrece se han utilizado principalmente las orientadas a un desarrollo mediante las capas modelo-vista-controlador.

En el archivo de configuración se definen los *beans* necesarios de spring, el nombre de los paquetes de Java, los dos “view resolver” uno para los JSP y otro para el informe en pdf, así como todas las asociaciones de las clases que implementaban las interfaces usadas.

En cada clase controlador, por ejemplo, se usan siempre las interfaces del modelo y en la línea anterior se especifica *@autowired*. Esto hará que spring busque en su archivo de configuración el *bean* que especifica la clase que implementa la interfaz.

Adicionalmente, spring se encarga de las conexiones a la base de datos. En el archivo de configuración especifica el driver de la base de datos, así como dónde está la información para realizar la conexión. Spring gestionará las conexiones.

Por último, Maven fue utilizado para resolver las dependencias que tiene el proyecto: spring y otras librerías de spring concretas, la api de JSP, el conector MySql y Json.

2 DESARROLLO

A continuación se definen los requisitos funcionales y no funcionales, riesgos y el diseño que fueron descritos tanto inicialmente como durante todo el proceso, ya que el uso de la metodología de desarrollo ágil hace que todos estos puntos varíen durante el tiempo.

2.1 REQUISITOS FUNCIONALES

Los requisitos funcionales que son expuestos, especialmente los casos de uso y el esquema de pantallas, corresponden a la configuración por defecto de la aplicación, dado que los permisos para que un rol pueda realizar una acción o no, pueden ser modificados.

2.1.1 Casos de uso

A continuación se examinarán algunos de los casos de uso básicos así como algunos de los más llamativos que incluirán diagramas de actividad.

- Relación entre actores y resumen de casos de uso:

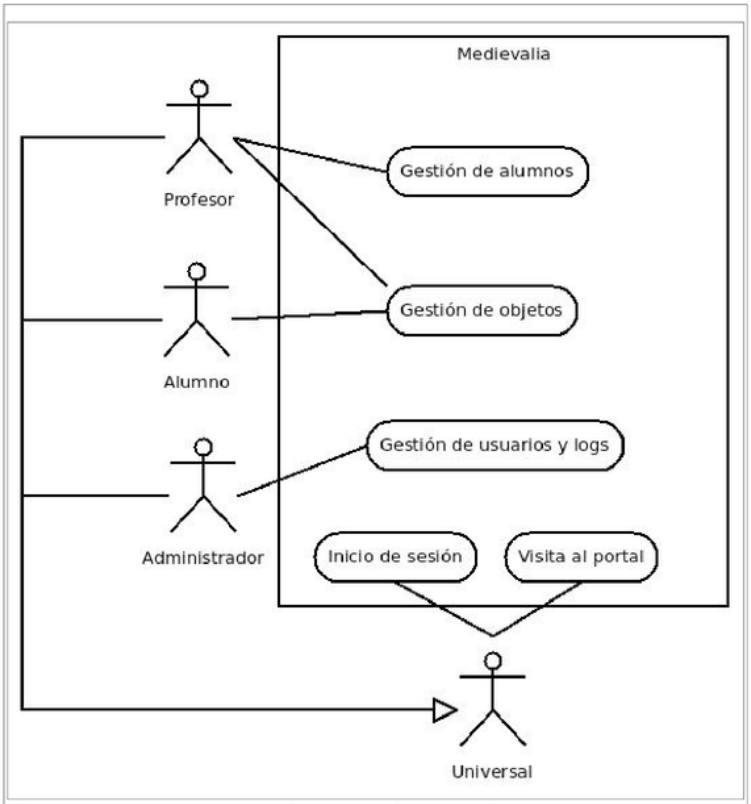

Ilustración 1: Resumen de casos de uso y herencia entre actores

- La pantalla de inicio de sesión tiene únicamente la opción de iniciar sesión.

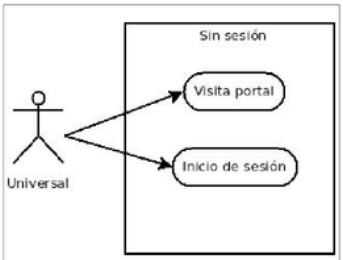

Ilustración 2: Diagrama de casos de uso de la pantalla de inicio

- La pantalla de inicio no contiene botones o enlaces, excepto para el rol profesor. El resto de acciones posibles se agrupan en la parte superior de cualquier pantalla, no solo de la pantalla de inicio, por lo que se especifican en la ilustración 3.

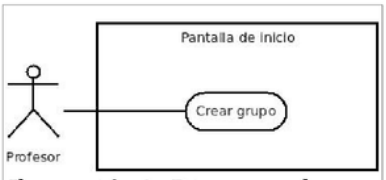

Ilustración 3: Diagrama de casos de uso de inicio

- Con sesión iniciada, los siguientes casos de uso son accesibles desde menú superior.

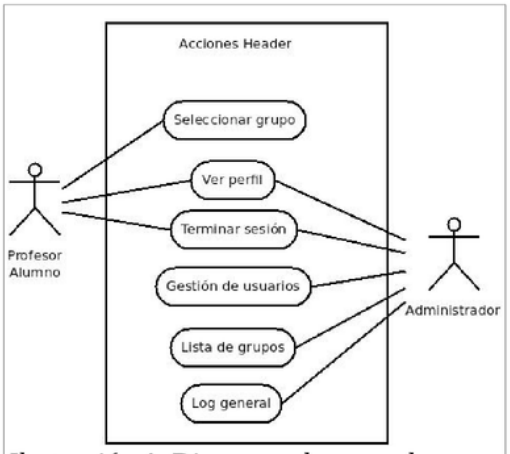

Ilustración 4: Diagrama de casos de uso para Header

- Una vez seleccionado un grupo, se puede visitar la página de matriculación con los siguientes casos de uso disponibles.

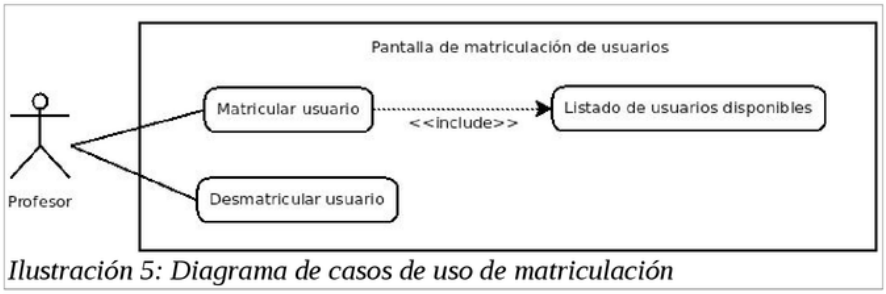

Ilustración 5: Diagrama de casos de uso de matriculación

- La pantalla de gestión de objeto es la parte más importante de la aplicación, donde se centran gran cantidad de casos de uso:

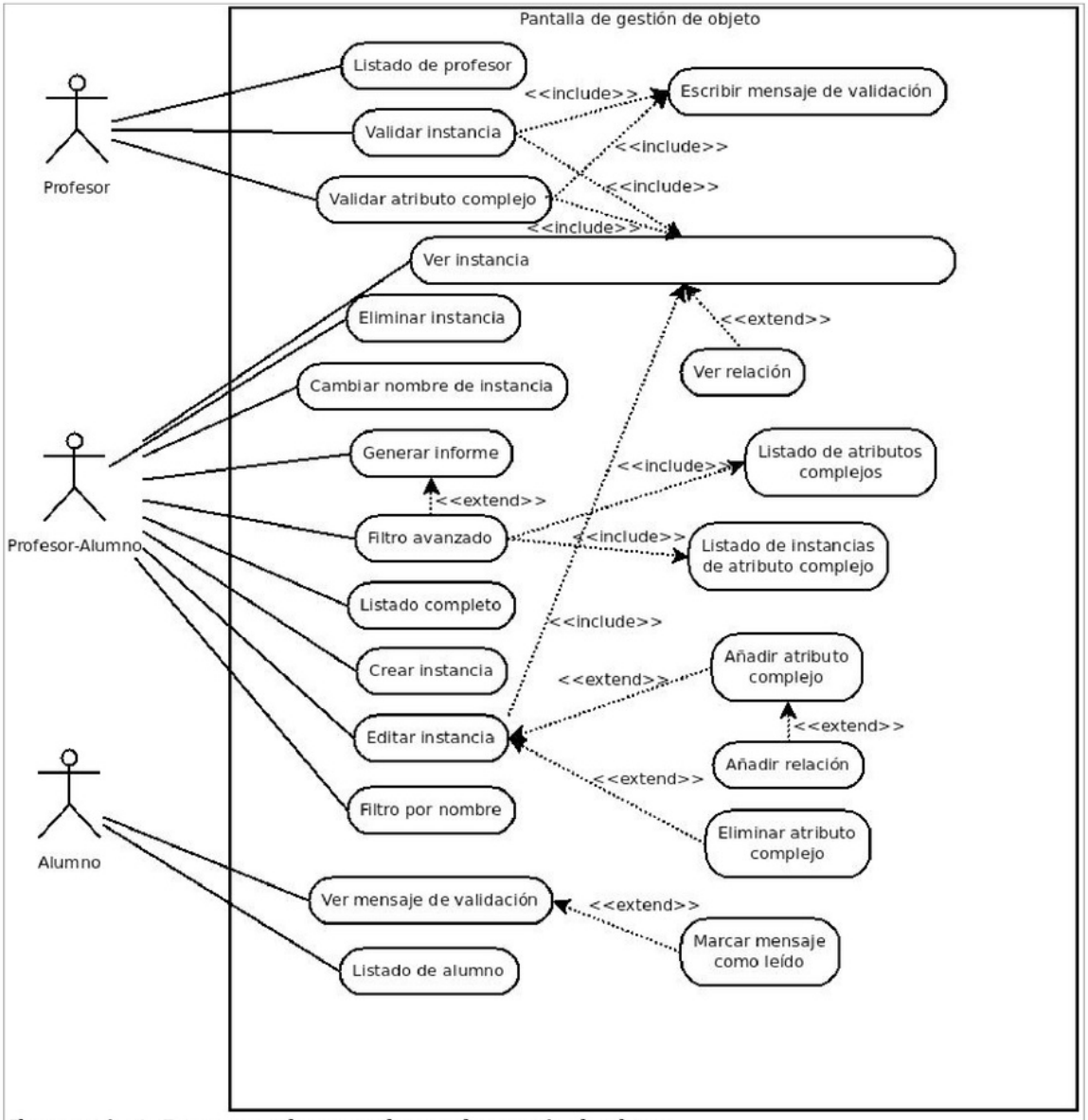

Ilustración 6: Diagrama de casos de uso de gestión de objeto

- El rol administrador en la pantalla de gestión de usuarios tendrá los siguientes casos de uso disponibles.

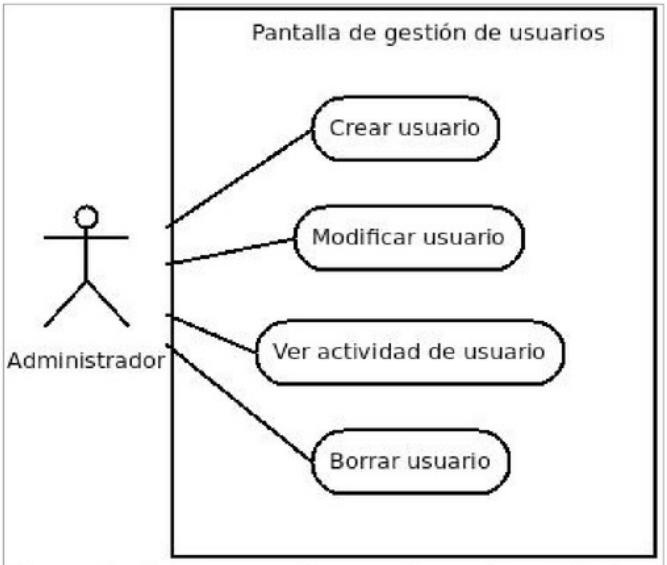

Ilustración 7: Diagrama de casos de uso de gestión de usuarios

- La pantalla de log general ofrece los siguientes casos de uso.

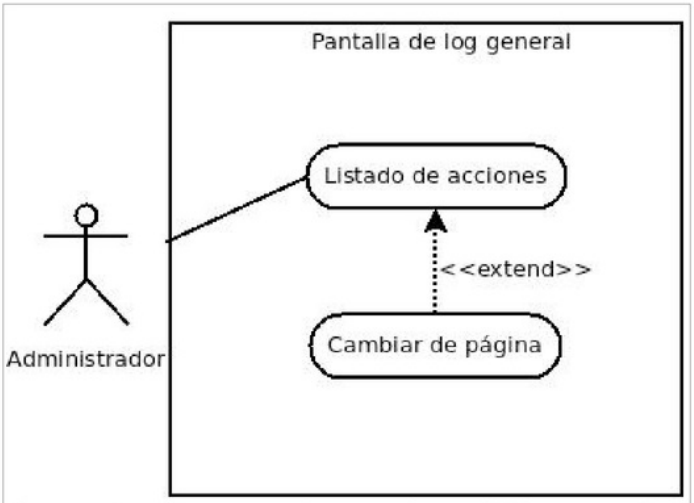

Ilustración 8: Diagrama de casos de uso del log general

- En la pantalla de listado de grupos el administrador únicamente tendrá el siguiente caso de uso.

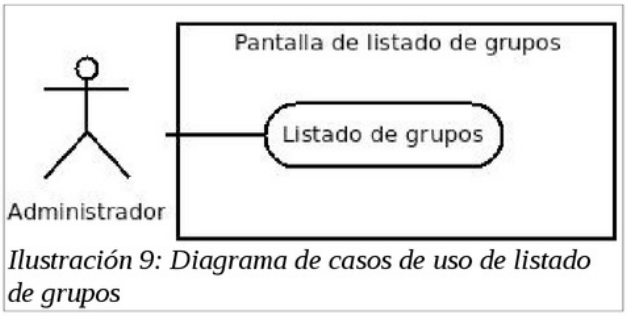

Ilustración 9: Diagrama de casos de uso de listado de grupos

- La pantalla de Gestión de contenido tiene los siguientes casos de uso:

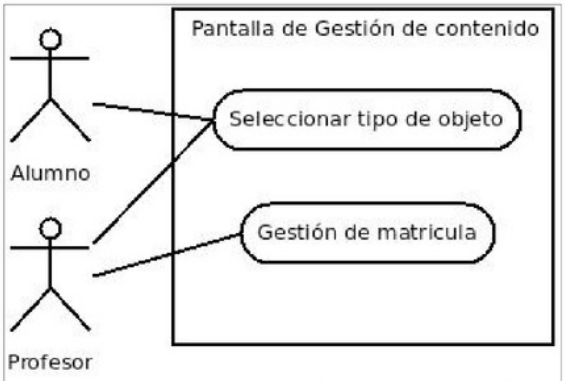

Ilustración 10: Diagrama de casos de uso de Gestión de contenido

- En la pantalla de perfil, las acciones son las siguientes:

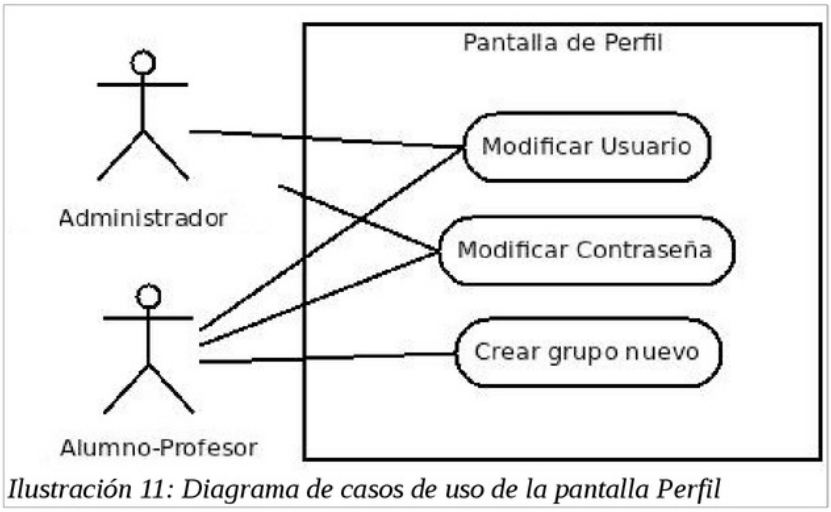

Ilustración 11: Diagrama de casos de uso de la pantalla Perfil

2.1.2 Esquemas de pantallas

Al tratarse de una aplicación web es posible realizar cualquier acción y potencialmente visitar cualquier pantalla escribiendo manualmente la URL. Sin embargo, las opciones que ofrece la aplicación a través de enlaces y botones, así como las restricciones de autorización permiten una navegación por la aplicación que viene recogida por los siguientes esquemas de pantallas, según los siguientes casos:

- Acceso al portal /medievalia

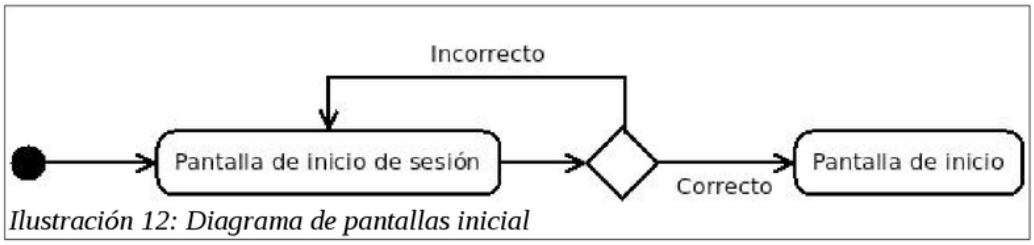

Ilustración 12: Diagrama de pantallas inicial

- Sesión iniciada con profesor

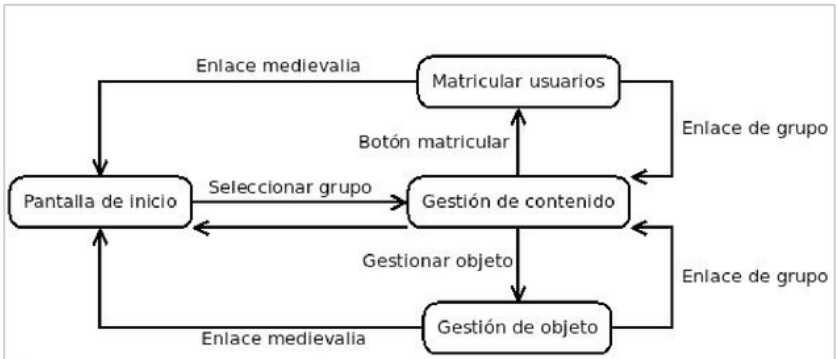

Ilustración 13: Diagrama de pantallas de profesor

- Sesión iniciada como alumno

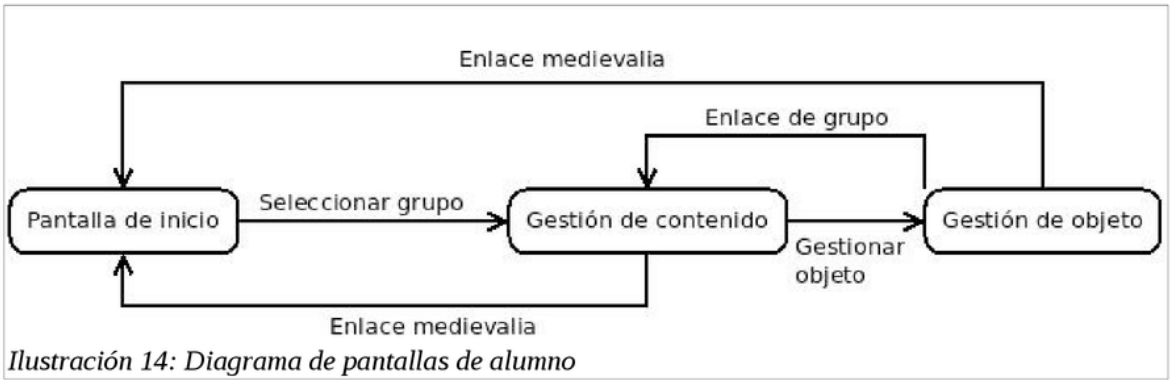

Ilustración 14: Diagrama de pantallas de alumno

- Peticiones desde el menú superior con administrador

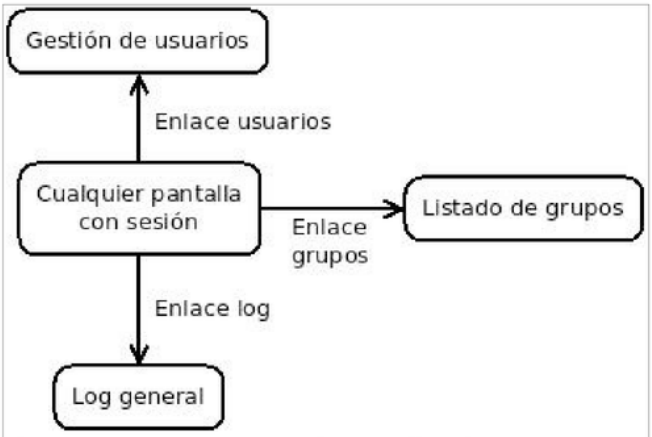

Ilustración 15: Diagrama de pantallas de administrador

- Peticiones desde el menú superior con cualquier rol

Ilustración 16: Diagrama de pantallas universal

Una vez identificadas las pantallas de la aplicación servirá para clasificar los diferentes diagramas de casos de uso y así localizarlos.

2.2 AUTENTICACIÓN Y AUTORIZACIÓN

Dado que en la aplicación es configurable el esquema, se consideró que otros aspectos fuesen también configurables. También es el caso de la autenticación y autorización.

Toda acción de usuario que se realice en la aplicación ha de estar autenticada y autorizada. Por ello no solo es necesario iniciar sesión para usar la aplicación, sino que cada acción requiere autorización y esta depende de la existencia de una tupla rol-acción que determina dicha autorización.

Las acciones que hay definidas son 53. Estas sí son fijas, ya que identifican ciertas funciones del código. Por otro lado, aunque hay 4 roles, pueden crearse nuevos roles.

idAction	action_name
0	Acción Nula
1	Login
2	Editar perfil
3	Lista de usuarios
4	Crear usuario

idRol	idAction
1	1
2	1
3	1
1	2
1	3
1	4

Ilustración 17: Ejemplos de acciones (izquierda) y tuplas rol-acción (derecha)

La lista de autorización son dichas tuplas y en la configuración por defecto tiene 67. Configurando la aplicación de una forma diferente habría que añadir o quitar tuplas de autorización. De hecho un nuevo rol sin tuplas no tendría permisos ni para iniciar sesión. Ese de hecho es el caso del rol inactivo, que responde a la necesidad de mantener no solo un histórico de usuarios antiguos, sino mantener datos como el creador de una instancia de objeto.

3 DISEÑO Y ORGANIZACIÓN DE LA APLICACIÓN

La aplicación web sigue el esquema básico modelo-vista-controlador al que añade una capa de persistencia y otros elementos auxiliares. A continuación se expone el funcionamiento general y en los apartados posteriores se entra en detalle de cada parte.

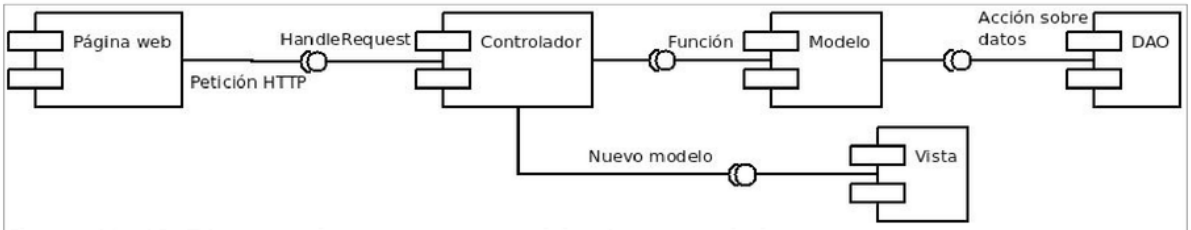

Ilustración 18: Diagrama de componentes modelo-vista-controlador

El diagrama anterior divide la aplicación en diferentes componentes genéricos que identifican paquetes o carpetas. Por un lado el usuario con el navegador tendrá una página web. En el lado del servidor hay 34 controladores asíncronos, así como otros 19 síncronos, que gestionan todas las peticiones. Estos, a su vez, utilizan las funciones que proveen los 7 servicios de la capa de Modelo. A su vez, estos servicios en algunos casos, no todos, han de acudir a la capa de persistencia para modificar los datos, y para ello hacen uso de los 6 DAO diferentes.

A modo de ejemplo, se enumera una serie de acciones genéricas para ilustrar el funcionamiento, para cada capa:

1. A través de una petición HTTP, realizada por el usuario, el sistema modelo-vista-controlador gestionado por Spring identifica el controlador solicitado por el usuario.
2. El controlador seleccionado comprueba que la petición es correcta a nivel léxico y sintáctico, realiza diferentes peticiones al modelo (autorización, cambios en datos, comprobaciones) y genera una vista nueva que es devuelta al cliente.
3. Cada elemento del modelo recibirá peticiones del controlador, analizará la semántica de la petición, comprobando que la petición es posible según el modelo descrito. Si es necesario consulta o actualiza la capa de persistencia usando uno o varios elementos DAO y devuelve la respuesta al controlador.
4. Un DAO recibe peticiones del modelo y realiza las operaciones necesarias para

modificar los datos persistentes. En el caso de MySql construye la petición y utiliza la conexión.

5. Las vistas reciben los datos necesarios ya procesados del controlador para construir una respuesta comprensible por el cliente en forma de HTML o mensajes JSON.

3.1.1 Diseño de la base de datos

La aplicación inicialmente se diseñó sobre una base de datos. Dicha base de datos era estática pero en esencia era una serie de tablas con relaciones para representar los objetos que tiene la instalación por defecto (Personaje, Cargo, Estudio, Dato...).

Posteriormente cuando se mejoró la aplicación para ser configurable y adaptarse así a cualquier ámbito de investigación cambió radicalmente de forma que, como se verá a continuación, no hay relación alguna con dicho modelo. El diseño de la base de datos actual comprende el siguiente modelo, expresado en su esquema físico de tablas.

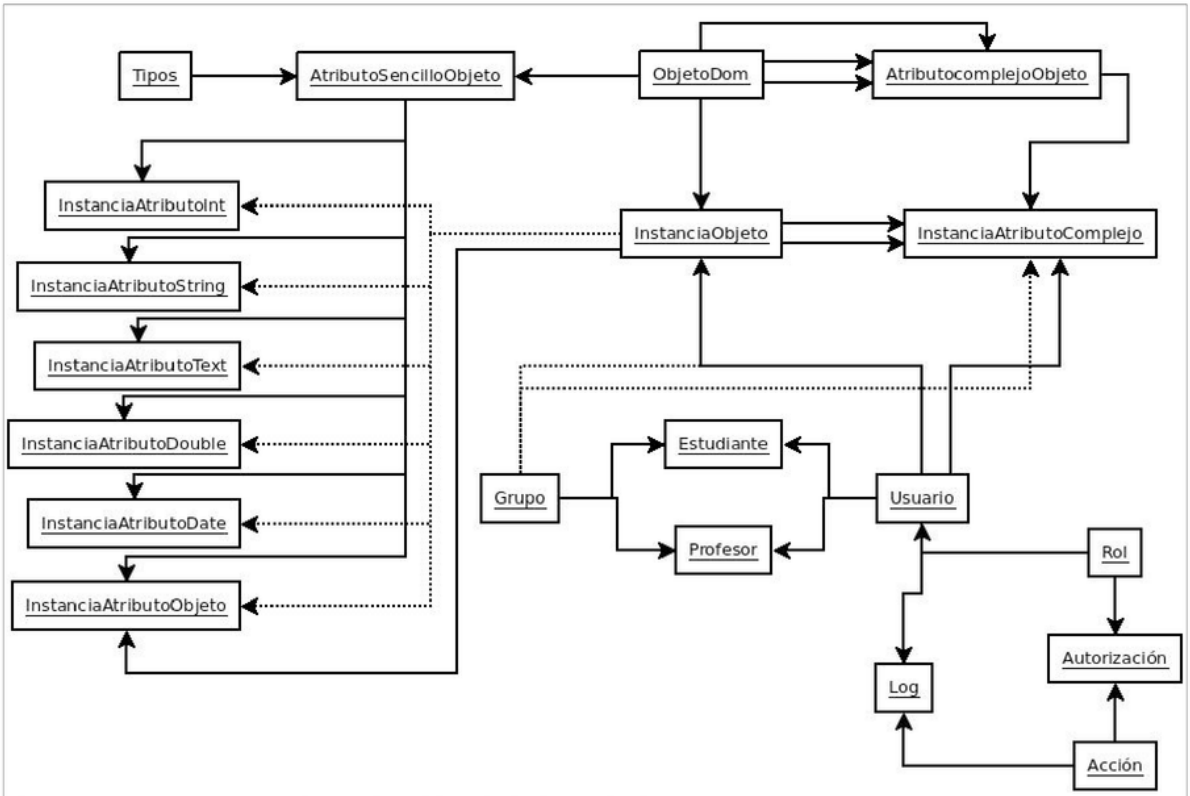

Ilustración 19: Esquema físico de tablas de la base de datos

La idea de este diseño se basa en las siguientes tres partes:

- Parte de infraestructura: las tablas de la parte inferior derecha son base del funcionamiento de la aplicación y son originarias de las primeras versiones de la aplicación. Las tablas son: Grupo, Estudiante, Profesor, Usuario, Log, Rol, Acción y Autorización. Con ellas se gestiona la aplicación en cuanto a autenticación, autorización, matriculación de usuarios y se detalla el log de actividad.
- Parte de definición de modelo. Con estas tablas se especifica los tipos de objeto, funcionando como una especie de pseudo-tablas y sus relaciones. Estas tablas una vez instalada la aplicación para un propósito específico no sería necesario modificarlas, pero cualquier cambio se refleja inmediatamente en la aplicación aunque el borrado está restringido en caso de existir instancias.
    - Cualquier objeto del dominio como estudio, cargo, lugar, personaje, etc, ahora será un objeto de dominio en **ObjetoDom**.
    - Cualquiera de estos objetos puede tener una lista de atributos sencillos de seis tipos diferentes, que son atributos de los cuales la instancia solo tiene uno de cada. En la tabla **AtributoSencilloObjeto** se almacenará cada tipo para cada **ObjetoDom**. La aplicación respeta estos tipos en su funcionamiento interno, aunque las claves ajenas impedirán incoherencias.
    - Por otro lado, también pueden tener atributos que sean complejos, esto es listas de otros **ObjetoDom**. Esto se controlará desde **AtributoComplejoObjeto** el cual tendrá dos referencias de **ObjetoDom**, una el objeto “padre” que será el que tenga una lista del objeto “hijo”.
- Parte de almacenamiento:
    - **InstanciaObjeto** es la clase central. Cada objeto podrá tener una serie de instancias creadas que serán almacenadas en esta tabla. Tiene además relación con el grupo al que pertenece y el usuario que lo crea.
    - **InstanciaAtributo[Int|String|...]** esta serie de tablas almacenarán un atributo por cada tipo de atributo sencillo para cada instancia. En el caso de ser objeto, tendrá

una referencia a **InstanciaObjetoDom**.

- **InstanciaAtributoComplejo**: contendrá las instancias de relación entre instancias de objetos. Por las claves primarias y ajenas usadas, se permite que una instancia de un objeto tenga varias instancias diferentes de otro objeto siempre y cuando esté reflejado en **AtributoComplejoObjeto**. La clave está en los cuatro campos de la clave primaria, cada cual es clave ajena de dos tablas simultáneamente. Además, también guarda relación con grupo y usuario al igual que **InstanciaObjeto**.

En el anexo I se detallan las tablas con todos sus campos, tipo de dato de cada campo, claves ajenas detalladas y claves primarias.

3.1.2 Objetos de dominio

El modelo de objetos sobre el que trabaja la aplicación viene definido por el siguiente diagrama:

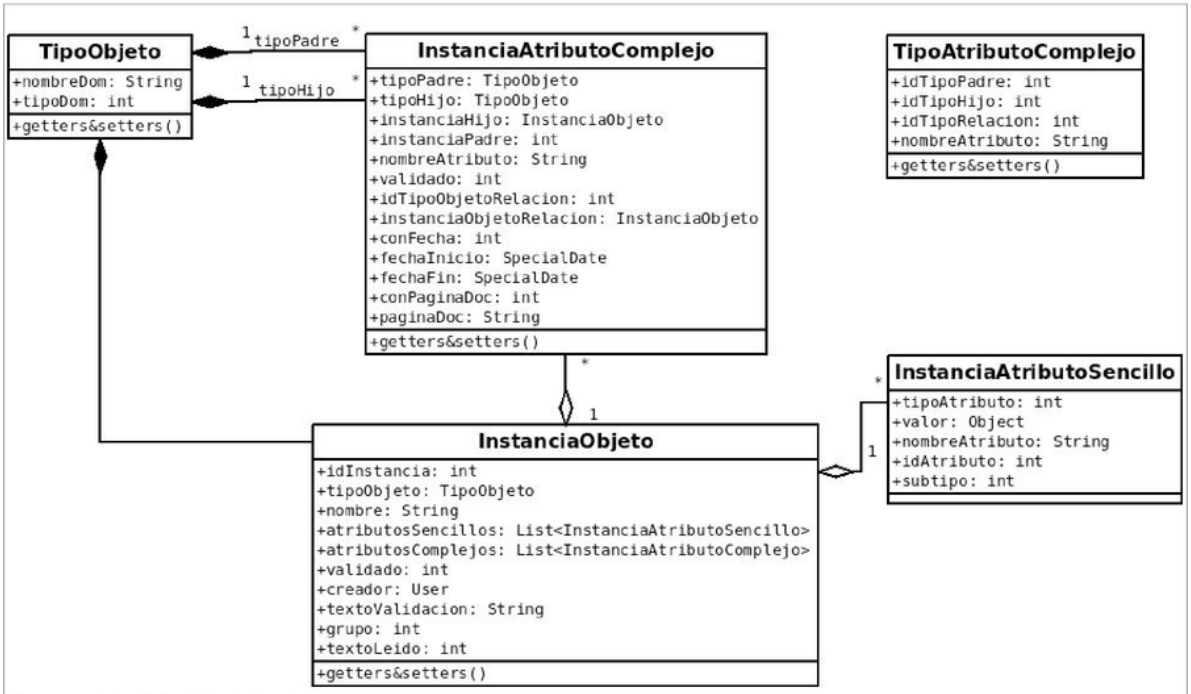

Ilustración 20: Modelo de clases

Hay otras clases auxiliares que no aparecen en el diagrama por ser simples reflejos de las tablas respectivas en la base de datos: Authorization, Group, Log, Role, Students, Teachers y

User.

3.1.3 Capa de Vista

La capa de vista está compuesta por los siguientes archivos:

- 16 archivos JSP que crean las páginas web que ve el usuario en el navegador a petición de los controladores síncronos. Están apoyados por:
  - 20 archivos js de JavaScript, propios de páginas concretas o generales, 1 archivo css general
  - Bootstrap: 6 css y 2 js
  - 1 archivo js de Jquery
- 4 archivos JSP comunes que son utilizados por los 16 JSP anteriores
- 16 archivos JSP que son utilizados para las comunicaciones asíncronas.

Toda página web que se muestre estará compuesta por elementos diferentes:

- **Cabecera**. El fichero header.jsp es común a todas las páginas. Construye según el usuario el menú superior e importa los archivos .js de cada página.
- **Cuerpo**. Son la mayor parte de los archivos JSP, propios de cada página.
- **Pie**. El fichero foot.jsp, al igual que el cabecero, es común.

Algunas de las funcionalidades JavaScript merecen mención especial dada la intensa implementación de funcionalidad mediante JavaScript, Jquery y Ajax.

- Las pantallas de matriculación de usuarios, perfil y gestión de objeto utilizan Ajax para, de forma asíncrona, crear, modificar o borrar objetos, con mensajes del resultado.
- La pantalla de gestión de objeto tiene funcionalidad de búsqueda de objetos, ocultando los elementos html de los objetos que no coinciden con la búsqueda dinámicamente. Permite mayor fluidez en la búsqueda, en vez de hacerlo en el servidor, más lento.
- La pantalla de gestión de objeto, adicionalmente, tiene un comportamiento más

complejo. De hecho cuenta con más de 1500 líneas de código JavaScript. Solo los botones relacionados con crear objeto, filtros e informe tienen funcionalidad una vez se realiza la carga de la página. Posteriormente se realizan una serie de acciones:

- Al finalizar la carga de la página se realiza dos peticiones asíncronas para cargar la lista completa y la lista correspondiente al rol: alumno o profesor.
- Una vez cargadas ambas listas se asigna función a los botones de cada objeto.

Pero más allá de este comportamiento, si el usuario quiere acceder a la información de una instancia concreta, ya sea ver o editar, se suceden otras acciones internas al funcionamiento de la página web mediante Jquery así como otras acciones asíncronas que se detallan en el siguiente diagrama de secuencia:

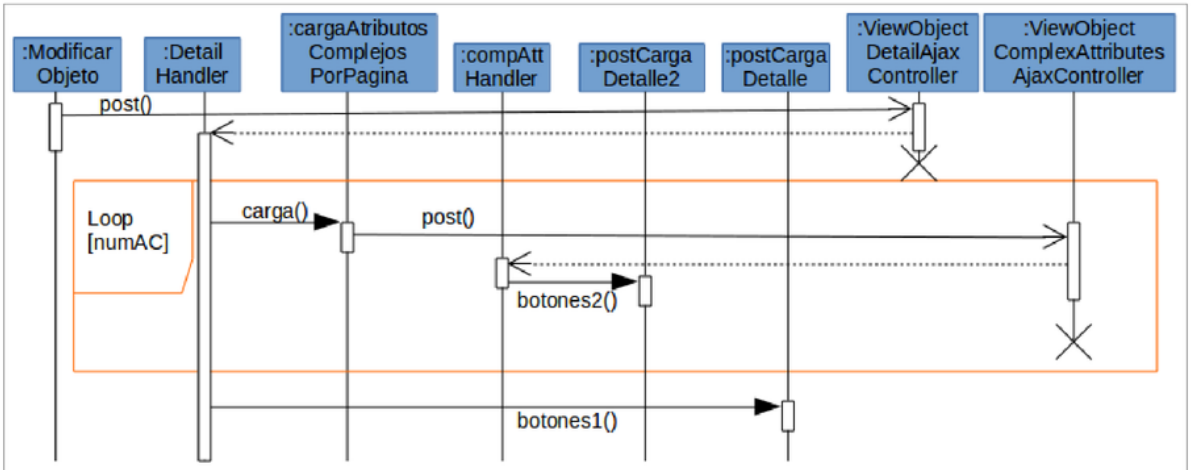

Ilustración 21: Diagrama de secuencia de la carga del detalle de una instancia

- Al ejecutar “modificar objeto” se envía una petición asíncrona (post() superior) con la carga de los datos de la instancia.
- Una función manejadora recibe la respuesta, crea la ventana con los atributos sencillos.
- La función anterior llama a la función de carga de atributos complejos por cada tipo de atributo complejo que haya recibido.
- Después la función manejadora otorga funcionalidad a los botones de la ventana, llamando a la función botones2().

- Mientras, la función de carga de atributos complejos, una vez por tipo, envía una petición asíncrona para recibir la lista de instancias de atributos complejos por tipo.
- Por cada respuesta, además de mostrar la lista de instancias de atributo complejo, otorga funcionalidad a los botones de cada elemento de la lista cargada llamando a botones2().

Esta funcionalidad permite la carga individualizada de cada tipo de atributo complejo por separado, para ser accesible en otros casos de uso, así como recargar fácilmente solo una parte de la información mostrada y no toda la ventana.

- Una última funcionalidad, muy extendida en toda la aplicación, es respecto a las peticiones asíncronas cuya respuesta es un mensaje sobre su resolución. En estos casos se analiza el mensaje y se muestra una ventana modal con el texto precargado. Estas ventanas existen previamente y están ocultas. Es usado decenas de veces en toda la aplicación.

3.1.4 Capa controlador síncrona

La capa controlador está dividida en dos grupos diferenciados por la forma de acceso. El primer grupo está formado por 17 controladores síncronos, que responden a las peticiones HTTP cuya respuesta es la página web que carga el navegador al completo. Siendo tantas se muestra a continuación un diagrama de clases de una de ellas:

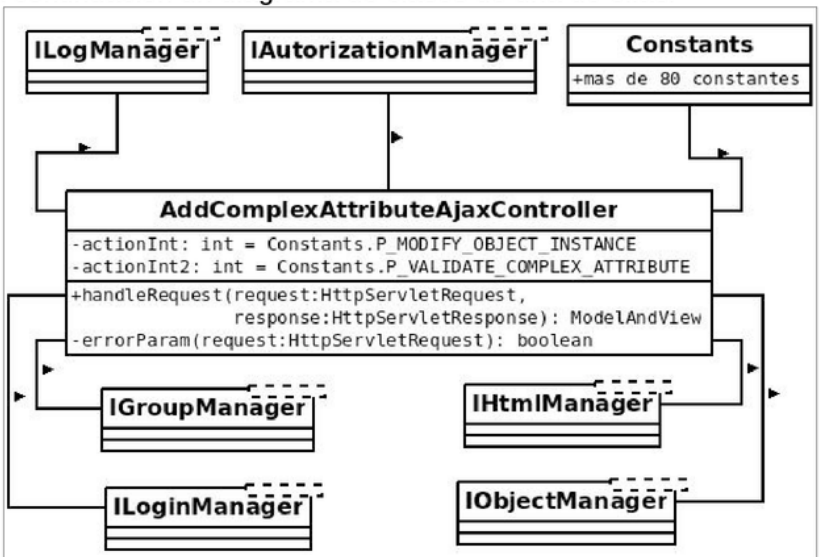

Ilustración 22: Ejemplo de controlador relacionado con clases de la capa de modelo y la clase Constants

Todos los controladores son similares, teniendo como atributos privados una serie de interfaces manager que serán implementadas por las clases del modelo.

Los controladores con sesión siempre realizarán las siguientes comprobaciones:

1. **Permisos y sesión**. Que el usuario de sesión tiene permisos para acceder. Si no hay datos de sesión como usuario, grupo u objeto, o no tiene permisos, devuelve una página de error.
2. **Parámetros**. Comprueba que son correctos usando el método errorParam().
3. **Modelo**. Realizan operaciones con la capa de modelo.
4. **Cabecera**. Una vez ha realizado las operaciones con los servicios del modelo, obtiene los elementos de la cabecera de la página web del servicio HtmlManager.
5. **Log**. Añade al log el resultado de la acción.
6. **Scripts**. Añade la lista de scripts necesarios para dicha página web.
7. **Vista**. Añade el resto de datos que la vista necesitará.

Las clases de cada controlador son las siguientes:

- **ContentManagerController**: es el controlador que genera la pantalla de Gestión de Contenido. Carga la lista de objetos del modelo e información de validaciones y número de instancias para cada usuario sobre cada objeto. Añade o cambia en la sesión el grupo seleccionado.
- **CreateUserController**: genera la pantalla de creación de usuario.
- **CreateUserController2**: crea el usuario según recibe parámetros y vuelve a la lista de usuarios. Es uno de los controladores más antiguos, por ello no se hizo asíncrono.
- **CuentasController**: muestra la página de la lista de usuarios.
- **DeleteUserController**: borra un usuario y realiza las mismas acciones que **CuentasController** para consruir la pantalla de usuarios. Al igual que los anteriores es antiguo y no es asíncrono.
- **GeneralLogController**: construye la página de log general con la primera página

cargada.

- **GroupController**: únicamente muestra una lista de grupos. Perdió funcionalidades y vistas de administrador, que lo solicitaban en un principio, en favor de otras vistas de profesores (además de pasar a funcionar de forma asíncrona).
- **GroupParticipants**: genera la página de matriculación, con los usuarios ya matriculados en dos listas.
- **InicioController**: construye la página de inicio. Además crea sesión si no existía y los parámetros usuario y contraseña son correctos.
- **LogoutController**: destruye la sesión y devuelve la página de inicio de sesión.
- **ModifyUserController**: carga la página con los datos de un usuario para modificarlos.
- **ModifyUserController2**: modifica un usuario por parte de un administrador. Es otro controlador antiguo, dado que esta acción podría ser asíncrona.
- **ObjectController**: sirve la página de Gestión de objeto, donde se ven todas las instancias de un tipo de objeto. Añade o modifica de la sesión el tipo de objeto seleccionado. No carga listas de instancias dado que esto se hace de forma asíncrona.
- **PdfController**: genera el archivo pdf de informe de objetos. Esta clase, además de obtener información del modelo, utiliza varios métodos para completar la información que normalmente obtiene (ya que en las vistas se va obteniendo información según es solicitada) y así entregarla completa a las clases que construyen documentos pdf, que se explicarán en detalle en el epígrafe 2.4.8.
- **ProfileController**: obtiene toda la información del usuario, no incluyendo el log, mostrándola en una página para su edición.
- **UserDetailController**: genera para un administrador la página del log de un usuario concreto, con las últimas 10 acciones cargadas.
- **WelcomeController**: únicamente añade al log la visita y muestra la página de inicio de sesión.

3.1.5 Capa controlador asíncrona

El segundo grupo de controladores son los encargados de las peticiones asíncronas, realizadas mediante Ajax y son la mayoría de los controladores, en total 34 clases. El ejemplo de la ilustración 22 también es válido para estos controladores.

La mayor parte de ellos realizan acciones similares a los controladores síncronos:

1. **Permisos y sesión**. En los asíncronos además es bastante habitual que, según sea la autorización, se realice una acción u otra, no únicamente denegarla.
2. **Parámetros**.
3. **Modelo**.
4. **Log**.
5. **Vista o JSON**. A diferencia de los controladores síncronos, estos pueden devolver una vista generada por un JSP, o construir una respuesta JSON que será tratada mediante Jquery en la página del navegador cargada previamente. En muchos casos simplemente devuelve un mensaje con el resultado de la operación, siempre en JSON, mostrándose un mensaje u otro en la pantalla.

Las clases del controlador son las siguientes:

- **AddComplexAttributeAjaxController**: solicita al modelo que añada una nueva instancia de atributo complejo a una instancia de objeto, con la información de la relación incluida si es pertinente.
- **CompleteObjectListAjaxController**: obtiene la lista de instancias de objeto completa del tipo seleccionado en sesión. Además, según sean los permisos del usuario, se indicará a la vista los botones que podrá usar el usuario.
- **CreateGroupAjaxController**: solicita crear un nuevo grupo al modelo.
- **CreateObjectInstanceAjaxController**: solicita crear una instancia del tipo de objeto seleccionado en sesión.
- **DeleteObjectAjaxController**: dependiendo de los permisos solicita al modelo eliminar una instancia cualquiera o únicamente instancias que pertenezcan al usuario.

Igualmente el tipo de dato es el seleccionado en sesión.

- **DeleteUserAjaxController**: solicita al modelo borrar un usuario.
- **EnrollUserAjaxController**: solicita al modelo matricular un usuario al grupo seleccionado en sesión, tras realizar múltiples comprobaciones del usuario que matricula y el que es matriculado.
- **FilteredObjectListAjaxController**: recibe los filtros de atributos complejos, aplica el filtro a la lista completa y la guarda en sesión. En adelante, la lista completa quedará filtrada hasta que se elimine o cambie el filtro. Está orientado a los informes.
- **GetComplexAttributeAjaxController**: recibe los datos que identifican a una instancia de atributo complejo (parte está en sesión) y solicita al modelo toda la información de dicha instancia, incluyendo la relación si la tiene.
- **GetComplexAttributeTypesAjaxController**: solicita la lista de tipos de atributo complejo que tiene un tipo de objeto.
- **GetEnrolledStudentsListAjaxController**: recoge la lista de usuarios matriculados como alumno al grupo seleccionado actualmente en sesión.
- **GetEnrolledTeachersListAjaxController**: recoge la lista de usuarios matriculados como profesor al grupo seleccionado actualmente en sesión.
- **GroupDirAjaxController**: pide al modelo la lista de grupos dirigidos por el usuario actual en la sesión.
- **GroupStudentAjaxController**: solicita la lista de grupos en los que el usuario actual de la sesión está matriculado como alumno.
- **GroupTeachAjaxController**: recoge del modelo la lista de grupos en los que el usuario actual en la sesión está matriculado como profesor.
- **ModifyUserAjaxController**: modifica el nombre de usuario o el nombre largo del usuario actual en la sesión.
- **ModifyUserPassAjaxController**: cambia la contraseña del usuario actual en la

sesión.

- **PossibleUserListToGroupAjaxController**: pide al modelo la lista de usuarios que no están matriculados en el grupo seleccionado en sesión, y la lista de roles.
- **RemoveComplexAttributeAjaxController**: solicita eliminar una instancia de atributo complejo identificada por los parámetros y el tipo seleccionado en sesión.
- **RenameObjectAjaxController**: renombra una instancia de objeto con la información de la petición y el tipo seleccionado en sesión.
- **SelectGroupAjaxController**: selecciona de forma asíncrona el grupo actual en sesión.
- **SetComplexAttributeTextReadedAjaxController**: cambia el estado del texto de validación a “leído”.
- **SimpleAttributeAjaxController**: cambia o da un valor a los atributos simples de una instancia de objeto, comprobando los tipos recibidos.
- **StudentObjectListAjaxController**: recoge la lista de instancias de objeto del tipo y grupo seleccionado en la sesión, que han sido creadas o modificadas por el usuario actual, teniendo este el rol alumno.
- **TeacherObjectListAjaxController**: recoge la lista de instancias de objeto del tipo y grupo seleccionado en la sesión que no han sido validadas por alumnos del grupo. También las que estando validadas tienen atributos complejos por validar.
- **UnenrollUserAjaxController**: desmatricula a un usuario alumno o profesor, dependiendo de los permisos.
- **UpdateComplexAttributeAjaxController**: actualiza la información de relación de una instancia de atributo complejo.
- **UserActivityAjaxController**: cambia la página del log de actividad de un usuario concreto de la aplicación.
- **UserGeneralActivityAjaxController**: cambia la página del log de actividad general de la aplicación.

- **ValidateComplexAttributeAjaxController**: cambia el estado a “validado” o no de una instancia de atributo compleja añadiendo un texto de validación.
- **ValidateObjectInstanceAjaxController**: cambia el estado a “validado” o no de una instancia de objeto, añadiendo un texto de validación.
- **ViewObjectInstanceComplexAttributesAjaxController**: carga la lista de instancias de atributo complejo de una instancia de objeto de un tipo de atributo complejo concreto indicado en la petición. También la lista de instancias de atributo complejo que están disponibles. Por lo tanto, para cargar el detalle de una instancia completa este controlador recibirá una petición por cada tipo de atributo complejo que tenga el tipo de objeto.
- **ViewObjectInstanceDetailAjaxController**: carga los atributos sencillos de una instancia de objeto, y los tipos de atributo complejo que tiene. Construye así una ventana con una pestaña con los atributos sencillos, y una pestaña por cada tipo de atributo complejo, realizando peticiones posteriores para las listas de cada uno.

3.1.6 Capa de modelo

La capa de modelo implementa la funcionalidad de la aplicación en cuanto a la gestión de los objetos de dominio y de gestión de la aplicación en general. En la ilustración 23 se representa las clases que lo implementan y sus relaciones.

Se parte de una serie de interfaces que definen toda la funcionalidad que tendrá la aplicación. Todas las clases de la capa controlador y modelo usarán interfaces y no las clases directamente. Spring Framework se encarga mediante el autoenlazado (*@autowired*) de concretar cuáles son las clases que implementan las interfaces, como se explica en el epígrafe 2.5.4.

Cada clase del modelo usará una única interfaz de la capa de persistencia con el mismo prefijo en el nombre, excepto HtmlManager que en esta versión no necesita persistencia alguna, pero como se detallará en el epígrafe 4.4 de futuras versiones, podría necesitarla para almacenar preferencias y hábitos del usuario.

A continuación se describe las clases y de qué se encarga cada una.

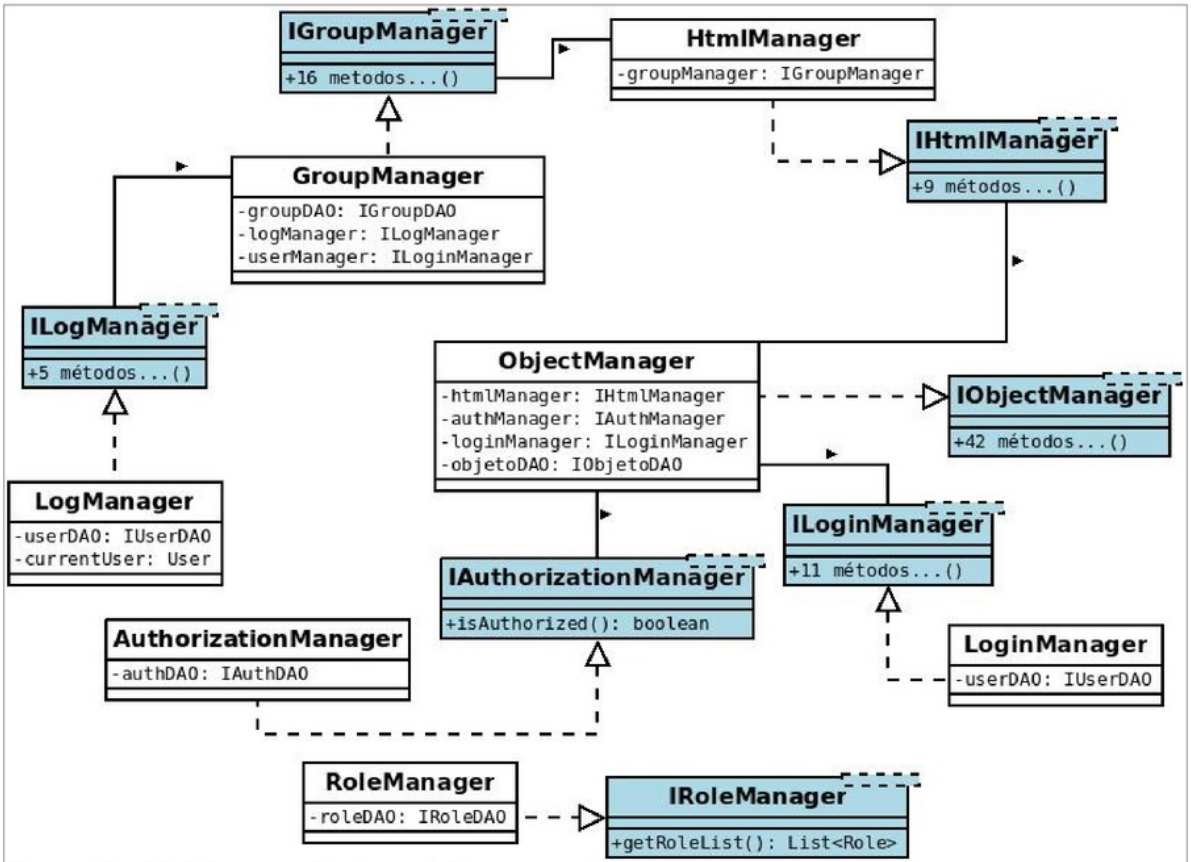

Ilustración 23: Diagrama de clases de la capa modelo

- **IRoleManager**: su funcionalidad sobre los roles es muy sencilla, ya que no se contempla la modificación de los roles desde la aplicación, sino directamente sobre la base de datos. Por ello únicamente devuelve la lista de roles.
- **ILogManager**: se encarga de escribir en el log todas las acciones realizadas en la aplicación, así como de posteriormente mostrarlas al administrador.
- **IGroupManager**: su función sobre los grupos abarca desde su creación, listado y matriculación, hasta lista de matriculados o lista de grupos de un usuario, siempre separando si es alumno o profesor.
- **IHtmlManager**: esta interfaz no va a tratar con los datos del modelo en si, sino con funciones propias de la interfaz que son usadas por todos los controladores. Se agrupó en una sola interfaz, aunque bien podría haber estado en el paquete de

utilidades descrito en 2.4.8 Otros componentes. Construye según la ocasión el menú de la parte superior de la pantalla, comprueba parámetros o construye mensajes de error de forma general según las peticiones sean síncronas, asíncronas o mediante JSON.

- **IObjectManager**: aquí se centra la mayor parte de la funcionalidad dado que en esta interfaz se trabaja con los tipos de objeto, atributos complejos, atributos sencillos así como todas sus instancias. Creación, listado, modificación, borrado, validación, comentarios, estadísticas, clasificaciones, relaciones, comprobaciones... Por ello acumula hasta 42 métodos, algunos de los cuales son usados múltiples veces en la clase que lo implementa o en muchos controladores. Además es la clase que usa más intensivamente los servicios de otras interfaces.
- **ILoginManager**: el nombre hace referencia al inicio de sesión, que fue su primera funcionalidad, pero abarca también acciones relacionadas con usuarios, como creación, modificación y borrado.
- **IAuthorizationManager**: la única funcionalidad que trata es la autorización de usuarios con acciones, según el rol del usuario. Las únicas autorizaciones que no trata la clase que la implementa son las que no están relacionadas con el rol, sino con el hecho de que un usuario esté matriculado como profesor o alumno.

3.1.7 Capa de persistencia

La capa de persistencia, de forma similar a la capa de modelo, son una serie de interfaces que describen toda la funcionalidad que necesita persistencia, y que es implementada por unas clases que específicamente la tratan mediante una base de datos MySql.

Como se puede observar en el diagrama de clases, el número de métodos de estas interfaces difiere del número de métodos de la clase que implementa el modelo y usa este DAO. Esto es debido a que algunos métodos son usados varias veces en el modelo o, al contrario, un método del modelo usa varios métodos diferentes de la capa de persistencia.

En el la ilustración 24 se observa esta diferencia.

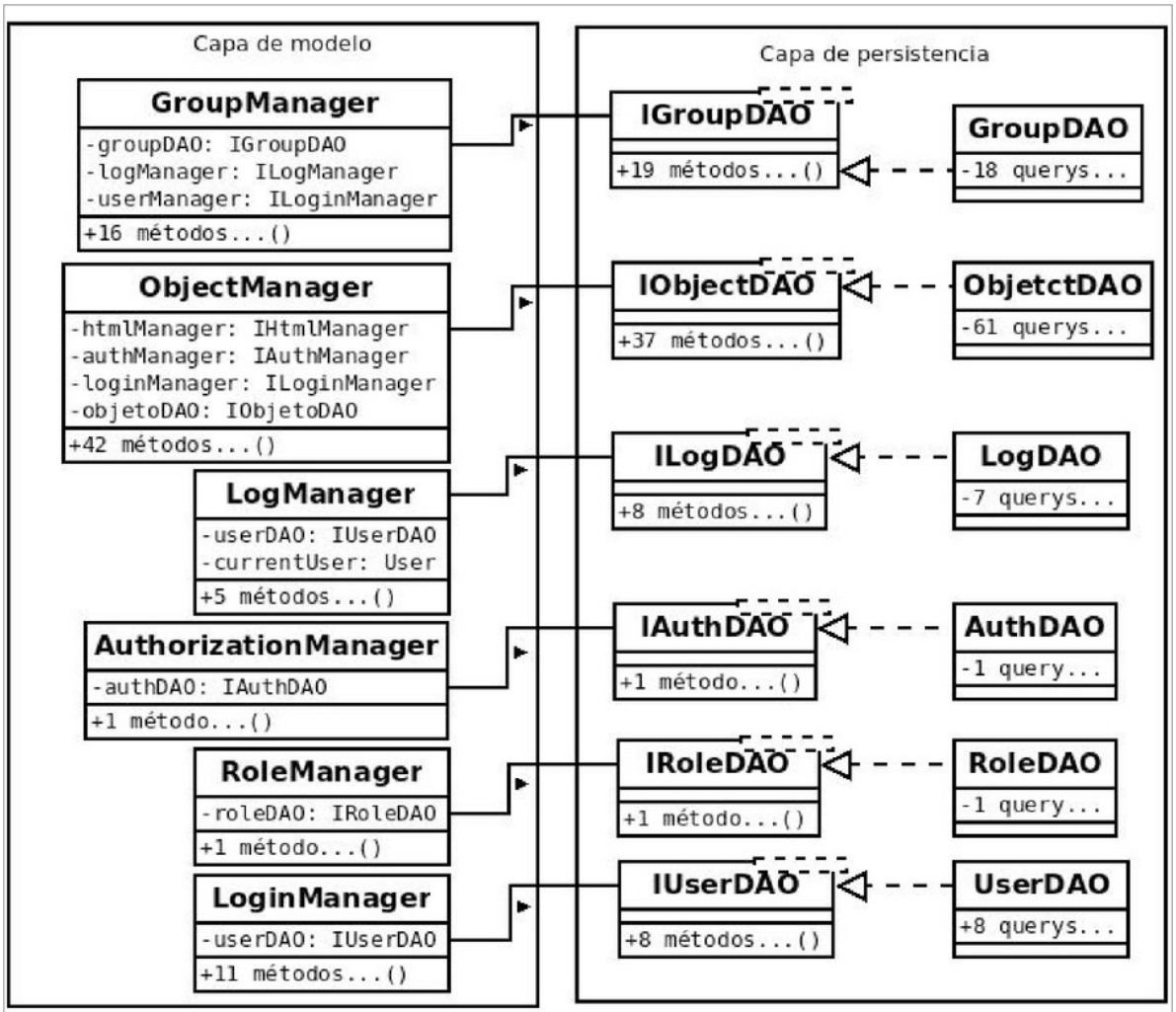

Ilustración 24: Diagrama de clases de la capa de persistencia

La herramienta que se usa para la comunicación con la base de datos es JDBC. Concretamente se ha utilizado la clase **JDBCTemplate**, que es otorgada por **Spring Framework**. De esta forma no se realiza manualmente la conexión, sino que se encarga Spring de ello. Como se detallará en el epígrafe 3.2 Guía de instalación, la dirección de la base de datos, el nombre del esquema, el usuario y la contraseña están almacenados en un fichero llamado “application.properties” y que es leído por Spring.

Se prefirió trabajar directamente con las *querys* directamente, ya que además de que hay clases que no están traducidas directamente a la base de datos, algunas consultas son complejas.

Cada clase DAO guarda como atributos privados las *querys* con los parámetros marcados con "?" para ser sustituidos al usarse. Esto, entre otras cosas, asegura que un usuario malintencionado no pueda realizar inyección sql para realizar acciones no permitidas**.**

A continuación se detalla parte de la funcionalidad de las clases DAO que no haya sido ya explicada en la capa de modelo.

- **IGroupDAO**: dado que GroupManager diferencia internamente entre usuarios matriculados como profesor o alumno, en muchos de sus métodos se emplean diferentes métodos de IGroupDAO, según se cumplan o no condiciones independientes de la autorización.
- **IObjectDAO**: de forma parecida que en la interfaz anterior, son comprobaciones que realiza IObjectManager que pueden dar lugar a diferentes métodos. Además de esto, IObjectDAO también tiene métodos que se utilizan de forma diferente según algunos atributos de los objetos a guardar. Guarda muchas más *querys* que métodos por el hecho de que hay diversas variantes de algunas, y no todas ellas no son *querys* completas, sino partes que son utilizadas para crear otras mayores, como es el caso de los filtros:
    - Según se recibe la lista de filtros con sus conjunciones, se añade a la *query* inicial las condiciones, una a una.
    - Cada condición es una única *query*, diferenciada si usa la conjunción AND o la conjunción OR. Es tratada de forma particular y sus parámetros son sustituidos antes de unirse a la *query* resultante.

  Otro punto a destacar es que el modelo utiliza un solo método para, por ejemplo, añadir un atributo sencillo. En ObjectDAO se discrimina según el tipo de atributo para realizar una *query* u otra, dado que son tablas diferenciadas.
- **IUserDAO**: en la clase que la implementa es donde se realiza la encriptación con AES de la contraseña, que es guardada encriptada en la base de datos. Cualquier acceso a la base de datos no revelará las contraseñas. La comparación de contraseñas para la autenticación es una comparación de las contraseñas encriptadas, en vez de desencriptar la guardada y compararla con la recibida.

3.1.8 Otros componentes

Otras clases son importantes aunque se no corresponden totalmente con las capas anteriores, ni tienen tanto peso o una función común como para representar una propia.

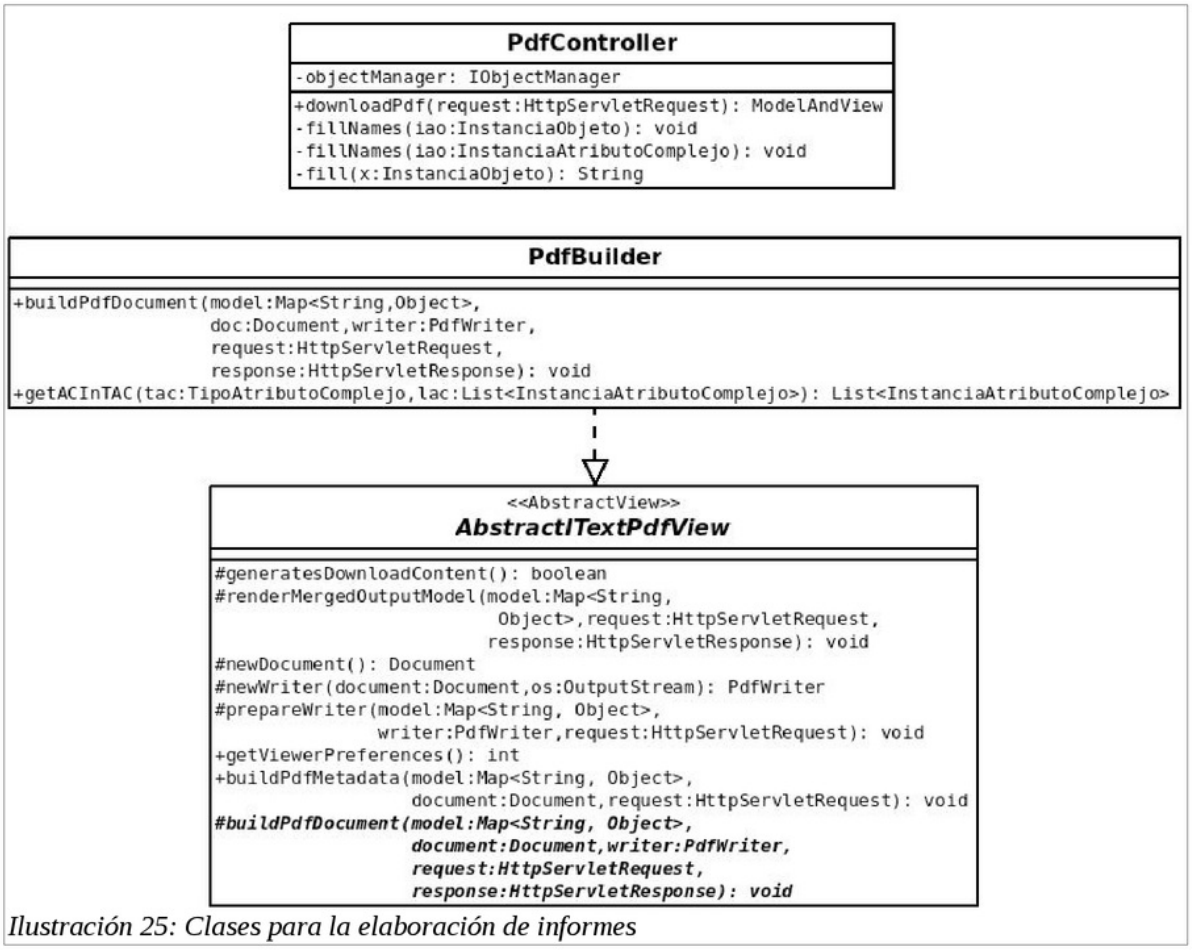

Ilustración 25: Clases para la elaboración de informes

- **Informes en pdf**. Hay dos clases que están relacionadas únicamente con un controlador, que es **PdfController**. Estas dos clases son **AbstractITextPdfView** y **PdfBuilder**. Su función es, recibiendo los datos a mostrar del controlador, construir un documento pdf utilizando para ello la librería **ITextPdf**. Cualquier objeto del dominio puede ser representado. Creará una tabla por cada instancia. En las primeras líneas aparecerán los atributos sencillos. Después, agrupadas por cada tipo de atributo complejo, la lista de instancias de atributo complejo con la información completa de la relación.

La vista de este controlador no es resuelta por Spring como las demás, sino que fue necesario configurar otro “viewResolver” alternativo al que es usado por el resto de los controladores. Es especificado en el archivo views.properties que hace referencia exclusivamente a la clase PdfBuilder, de forma que el modelo para construir la respuesta llega a esta clase y no a un JSP.

- **Fecha especial**: hay dos motivos por los que se ha decidido implementar una clase específica para el manejo de fechas que imposibilitan usar las clases Data de Java como los tipos de datos MySql para fechas:
  - Dificultad o imposibilidad de utilizar fechas muy antiguas, como pueden ser 1500 a.C de aplicarse en historia antigua.
  - Manejo de imprecisiones. Hay eventos de los que se conoce día, mes y año. Otros en los que solo el mes y el año. Finalmente otros en los que solo el año. Por ello debe existir la posibilidad de almacenar fechas con estas imprecisiones.

  Siendo así, se optó por almacenar en la base de datos día, mes y año como campos separados, y crear un objeto que contemple estas restricciones y compruebe que no hay problemas. Por ello, tendrá la siguiente funcionalidad:
  - Tratamiento de nulos: impedir que si no se conoce el mes, se conozca el día y que si no se conoce el año, tampoco se conozcan el mes y el día.
  - Analizar de forma centralizada el mensaje HTTP request para obtener los datos día, mes, año, de forma uniforme.
  - Mostrarlo: de forma uniforme, aunque actualmente solo en formato dd-mm-aaaa.
  - Comprobación: impedir meses o días no válidos, incluyendo días como el 29 de febrero en años bisiestos o no bisiestos.

No fue necesario utilizar otros calendarios que el gregoriano, dado que según el procedimiento de investigación histórica, todas las fechas no gregorianas (julianas por ejemplo) son convertidas al gregoriano por el investigador al ser almacenadas.

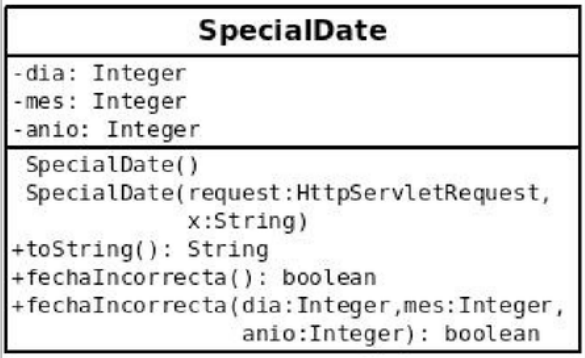

Ilustración 26: Clase para manejo de fechas SpecialDate

- **Constantes**. Se centralizaron todas las constantes que usa la aplicación. Entre los valores almacenados destacan:
  - Identificadores de acciones. La tabla de acciones no será modificada, ya que está ligada a la funcionalidad existente en la aplicación, y es útil tener una copia de los identificadores sin tener que acceder a la base de datos.
  - Valores numéricos. Algunos valores por defecto son descritos aquí, como el que representa “validado” y “no validado”, o valores mínimos y máximos de campos.
  - Valores textuales. Texto por defecto que se almacena en los campos de la base de datos, como el comentario de una instancia creada por un profesor.
- **Header**. Cada elemento almacena una url, un nombre de servicio y un nombre a mostrar y una lista de Header, de forma que se puede crear una estructura de árbol. Es utilizado para los menús de la parte superior de las páginas, aunque en la vista header.jsp no se implementa de forma recursiva, sino solo hasta segundo nivel.
- **ListaAtributoSencillo**: se usa puntualmente para construir los desplegables con las diferentes opciones de un atributo sencillo cuando es tipo objeto, por ello guarda una lista de valores posibles de instancias para un tipo de objeto.
- **ListaRelaciones**: similar a ListaAtributoSencillo, pero en este caso para relaciones.

4 IMPLEMENTACIÓN

4.1 REQUISITOS DEL SISTEMA

Para un correcto funcionamiento de la aplicación será necesario:

- Un servicio Apache Tomcat 7.0 o superior, en un servidor que pueda soportarlo y con acceso a la red con el alcance que sea deseado (intranet o, más habitualmente, internet). Es recomendable que dicho servidor tenga un nombre DNS asociado para ser accedido de forma más sencilla.
- El servidor ha de tener instalada un entorno de ejecución de Java 1.8 o superior. Además será necesario que en las librerías de Tomcat esté presente el conector de MySql para Java.
- Una base de datos MySql accesible desde el servidor.
- Los usuarios finales han de disponer de un navegador y acceso por red al servidor, ya sea desde un PC, tablet o teléfono móvil. Los navegadores pueden ser Firefox, Edge y Chrome en sus últimas versiones, Internet Explorer desde la versión 8 y Safari[1]. El ancho de banda no es significativo, pero sí mejorará la experiencia de usuario si es alto.

4.2 GUÍA DE INSTALACIÓN

Para instalar la aplicación en un servidor y que sea completamente funcional se han de seguir los siguientes pasos.

1. Disponer del archivo medievalia.war
2. Instalar un servicio Apache Tomcat 7.0 o superior. Se ha de tener acceso como administrador o al menos contacto con el administrador.
3. Instalar un servicio MySql, o tener un esquema de base de datos con un usuario con todos los permisos sobre el mismo.
4. Editar el fichero dentro del archivo medievalia.war /WEB-INF/classes/application.properties y especificar los parámetros de conexión a la base

1 Ver más información en la documentación de Bootstrap en el epígrafe 5 de bibliografía, sobre los requisitos del navegador.

de datos: acceso al servidor, nombre del esquema, nombre del usuario y contraseña.

5. Añadir a las librerías de tomcat el conector de MySql. En sistemas POSIX generalmente se encuentra en el directorio /usr/share/tomcat/lib
6. Acceder a la base de datos e importar el archivo BDmedievalia.sql. Es una operación muy sencilla tanto desde comandos como accediendo desde phpMyAdmin.
7. Acceder al manager de Tomcat y desplegar el archivo medievalia.war
8. Acceder a la url del servidor y a la aplicación Medievalia, que si el servicio no ha cambiado el puerto estándar es: servidor:8080/medievalia
9. Introducir el usuario por defecto: “admin”; y la contraseña inicial “adminMed”.

Una vez se accede a la página web del punto 8 se confirma que la aplicación se ha desplegado con éxito en el servidor web.

Igualmente, si se inicia sesión con el usuario admin como indica el punto 9 y aparece la ventana inicial, se confirma que la conexión a la base de datos es correcta, dado que el inicio de sesión depende de esta.

Por último, para comprobar que el esquema se ha cargado correctamente, será necesario crear un usuario profesor e iniciar sesión con el mismo. Se ha de crear un grupo y seleccionarlo en el menú superior. De aparecer una ventana como se visualiza en la ilustración 28 se confirmará que el esquema se creó correctamente, y con ello, la aplicación se instaló con éxito.

4.3 Guía de cambio de esquema

El esquema que se despliega por defecto al instalar la aplicación es de uso particular en la investigación histórica. Responde a un esquema entidad-relación como el siguiente.

Si la instalación se va a realizar para otro campo de investigación, o se quiere modificar la existente, se debe comprender cómo se configura. Inicialmente hay dos formas de actuar:

Con la primera, si se tiene conocimiento de bases de datos, se debería realizar un esquema entidad-relación que interprete los requerimientos del área específica. Bien es cierto que el esquema resultante probablemente no pueda representar cualquier esquema entidad-

relación, pero sí lo hará en gran parte de los casos.

Es posible también que si se ha usado un tiempo la aplicación Medievalia o similar, y se comprenden los conceptos de tipo de objeto, instancia de objeto, atributo sencillo e instancia de atributo sencillo, y atributo complejo y su instancia, se pueda hacer un diseño de otro esquema, sin que sea entidad-relación y por lo tanto sin conocimientos en bases de datos.

Una vez se tiene claro el diseño se han de añadir registros en ciertas tablas de la base de datos, que se verán en los siguientes apartados. Para ilustrar todo esto se verá a su vez cómo está en la instalación por defecto y un ejemplo de cambios posibles y sus resultados.

4.3.1 Tipos de objeto

Los tipos de objeto serían los equivalentes a una tabla en MySql, o a una entidad en entidad-relación. En este punto se ha de saber únicamente cuántas hay y su nombre, no sus atributos.

Por cada una de ellas se añadirá un registro a ObjetoDom, indicando el nombre en el campo “nombreObjeto”. No es necesario indicar un número en idObjeto ya que es autonumérico. Así, en la aplicación original se tiene 12 objetos por defecto, como muestra la ilustración 27.

idObjeto	nombreObjeto
1	Cargos
2	Estudios
3	Personajes
4	Lugares
6	Temas
7	Subtemas
9	Autores
10	Datos
11	Documentación
12	Archivos

Ilustración 27: Tabla ObjetoDom

Si se añade a la tabla ObjetoDom por defecto un registro con idObjeto 13 y un nombre “Ejemplo”, se crearía un nuevo objeto el resultado aparece en la ilustración 28.

Dicho objeto no tendría ningún atributo. Se pueden crear instancias de ese objeto, pero solo tendrán nombre, como las instancias de Cargo.

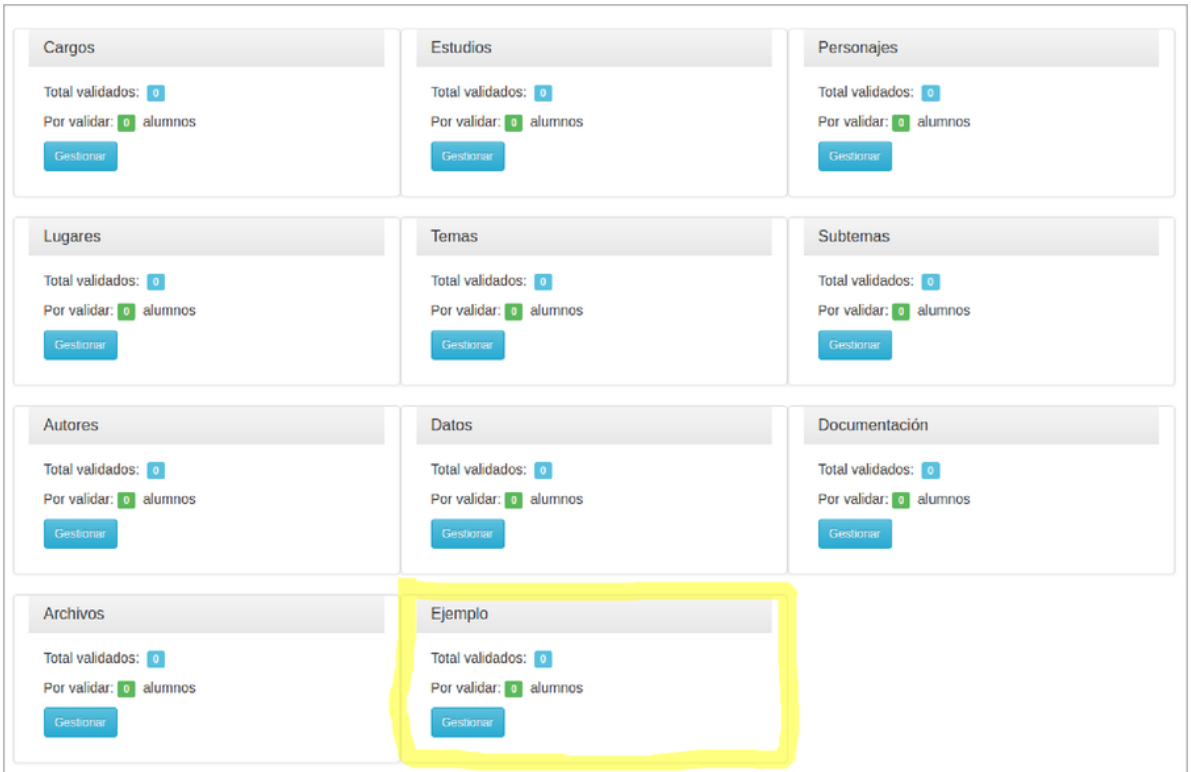

Ilustración 28: Resultado de introducir un registro a la tabla

4.3.2 Atributos sencillos

Los atributos sencillos son los que toman un único valor en cada instancia de objeto. Para añadir un atributo a un objeto, se insertará un registro en la tabla con los siguientes valores en cada campo:

- idAtributo: se ha de indicar el órden del atributo para cada objeto. Esto es, todos los objetos con un atributo sencillo tendrán un atributo con id 1. Si por ejemplo al objeto “Personaje” que tiene 5 atributos, numerados del 1 al 5, si se añade uno más deberá ser el 6.
- idObjeto: es el identificador del objeto que tiene el atributo sencillo. Si, por ejemplo, se quiere añadir a personaje un atributo sencillo, personaje figura en la tabla ObjetoDom como el 3, por lo tanto el número a insertar es el 3.
- nombreAtributo: es el texto que describirá la función del atributo y como se mostrará

en la interfaz.

- tipo: indicará qué tipo de atributo sencillo es. Si hay dudas se consulta la tabla Tipos:
  - 1: date. Campo fecha. Se mostrará siempre con día, mes y año en campos separados.
  - 2: double. Es un campo numérico real, que acepta decimales.
  - 3: int. Otro campo numérico entero, solo números sin decimales.
  - 4: string. Es un campo de texto no largo en exceso, hasta 255 caracteres.
  - 5: text. Otro campo de texto, pero hasta 65535 caracteres.
  - 6: objeto. Este campo será un desplegable cuyas opciones serán todas las instancias de un tipo de objeto concreto.
- subtipo: en caso de que el tipo sea 6, indicará el tipo de objeto del que mostrará las opciones disponibles. Si no es tipo 6 se ignorará, pero por convención se dejará como valor nulo.

Sabiendo la función de cada campo de dicha tabla, se mostrará un ejemplo del resultado de añadir a Personaje dos atributos sencillos.

1. idAtributo: 6, idObjeto: 3, nombreAtributo: “Ejemplo sencillo 1”, tipo: 4, subtipo: null

2. idAtributo: 7, idObjeto: 3, nombreAtributo: “Ejemplo sencillo 2”, tipo 6, subtipo: 11

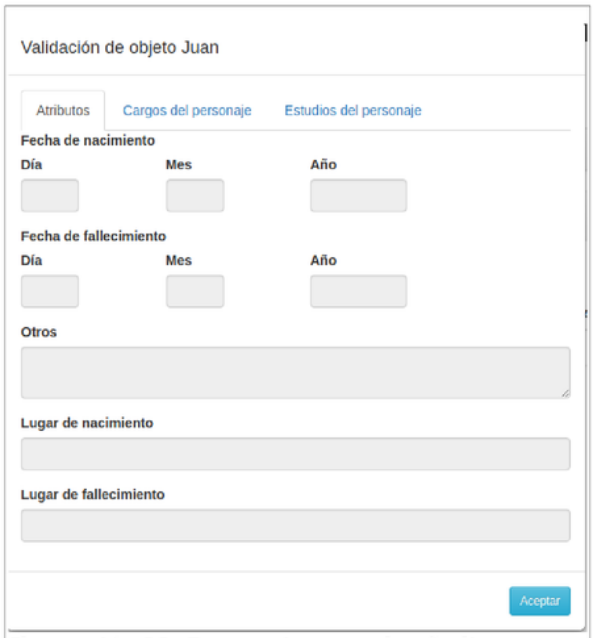

Ilustración 29: Personaje antes de añadir atributos sencillos

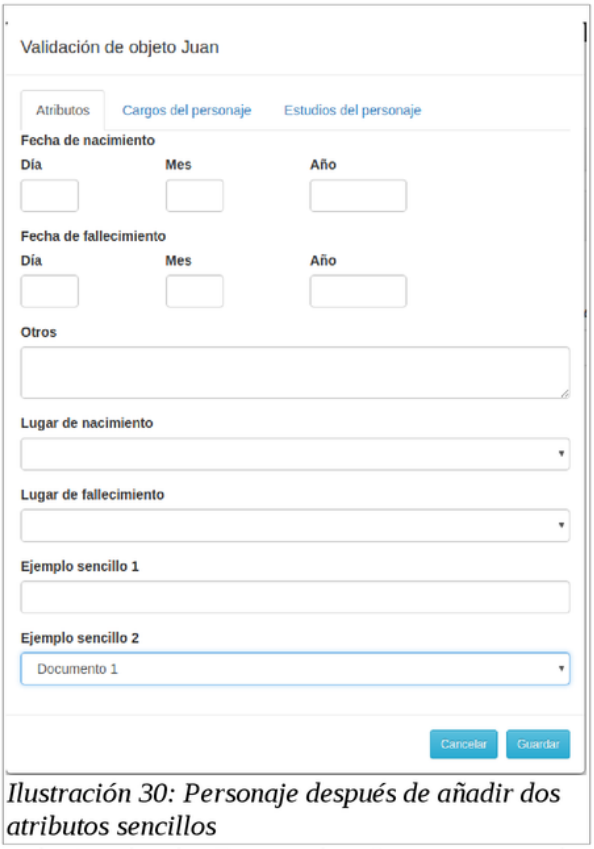

Ilustración 30: Personaje después de añadir dos atributos sencillos

Las imágenes corresponden al detalle de una instancia de Personaje. Como se puede observar, se han añadido dos campos más. En el primero se podrá escribir un texto. En el segundo hay un desplegable, cuyas opciones son todas las instancias validadas de tipo Documento (en la tabla ObjetoDom tiene id 11).

4.3.3 Atributos complejos

Los atributos complejos son aquellos que toman una lista de valores en cada instancia de objeto. Además, estos valores son necesariamente otras instancias de objeto. Por ejemplo, “Personaje” tiene dos atributos complejos: “Cargos de personaje” y “Estudios del personaje”. En “Cargos de personaje” por ejemplo, una instancia puede tener asignados “Rey”, “Cardenal” e “Infante”, siendo estos instancias de “Cargo”. Lo mismo sucedería con “Estudios del personaje” pero con instancias de “Estudio”.

La tabla AtributoComplejoObjeto es la encargada de recoger esta configuración y con el

esquema por defecto tiene el siguiente contenido:

idObjetoPadre	idObjetoHijo	NombreAtributo	idObjetoRelacion	conFecha	conPaginaDoc
3	1	Cargos del personaje	11	1	1
3	2	Estudios del personaje	11	1	1
10	3	Implicados	11	0	1
10	4	Localizaciones	11	0	1
10	6	Tema asociado	0	0	0
10	7	Subtema asociado	0	0	0
10	11	Documentación	11	0	1
11	4	Lugar asociado	0	0	0
11	9	Autor	0	0	0

Ilustración 31: Contenido de la tabla AtributoComplejoObjeto

Para añadir un atributo complejo a la tabla, se ha de añadir un registro a la tabla AtributoComplejoObjeto y primero se ha de comprender la función de cada campo:

- idObjetoPadre: es el identificador del objeto que tiene el atributo complejo. Si se añade "Tema" (6) a "Personaje" (3), será el 3.
- idObjetoHijo: es el identificador del objeto que es el atributo complejo. Si se añade "Tema" (6) a "Personaje" (3), será el 6.
- nombreAtributo: será el nombre que aparecerá en la interfaz para que el usuario identifique lo que se guarda en él.
- idObjetoRelacion: similar a idObjetoHijo, pero será el tipo de objeto que se asocia a la relación. Al añadir una instancia de atributo complejo, si este valor es 0, se añadirá automáticamente. Si el valor es un identificador de un tipo de objeto, se pedirá que se añada una instancia de este tipo de objeto. Su uso suele ser para la "bibliografía" en la que se basa el hecho de asignar el atributo complejo. En el esquema por defecto es 11.
- conFecha: como con idObjetoRelación, al añadir una instancia de atributo complejo, puede añadirse una fecha de inicio y una fecha de fin si el valor es 1.
- conPaginaDoc: es el último campo relacionado con las relaciones. En este caso, si el

valor es un 1, se añadirá un campo de texto a la relación. En el esquema por defecto se utiliza para guardar referencias de páginas, como “12-16” o “Volumen II, página 25”. Si es 0, no existirá dicho campo.

A continuación se muestra el ejemplo de insertar dos atributos complejos, insertando en la tabla indicada los dos siguientes registros.

1. idObjetoPadre: 3, idObjetoHijo: 9, nombreAtributo: “Ejemplo complejo 1”, idObjetoRelacion: 0, conFecha: 0, conPaginaDoc: 0.
2. idObjetoPadre: 3, idObjetoHijo: 4, nombreAtributo: “Ejemplo complejo 2”, idObjetoRelacion: 11, conFecha: 1, conPaginaDoc: 0.

El resultado al editar una instancia de personaje cualquiera es el siguiente:

Ilustración 32: Personaje antes de añadir los atributos complejos

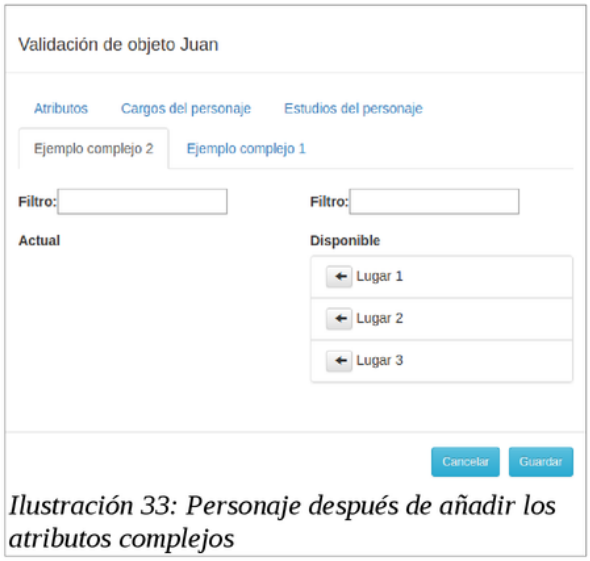

Ilustración 33: Personaje después de añadir los atributos complejos

Como puede observarse han aparecido dos pestañas más. En la imagen de la derecha aparece seleccionada la primera, “Ejemplo complejo 2”, que se le asignó en idObjetoHijo el identificador de “Lugar”, por ello aparecerá la lista de instancias de “Lugar”. Si se añade una instancia, la asignación tendrá información de la relación, mostrándose la ventana siguiente:

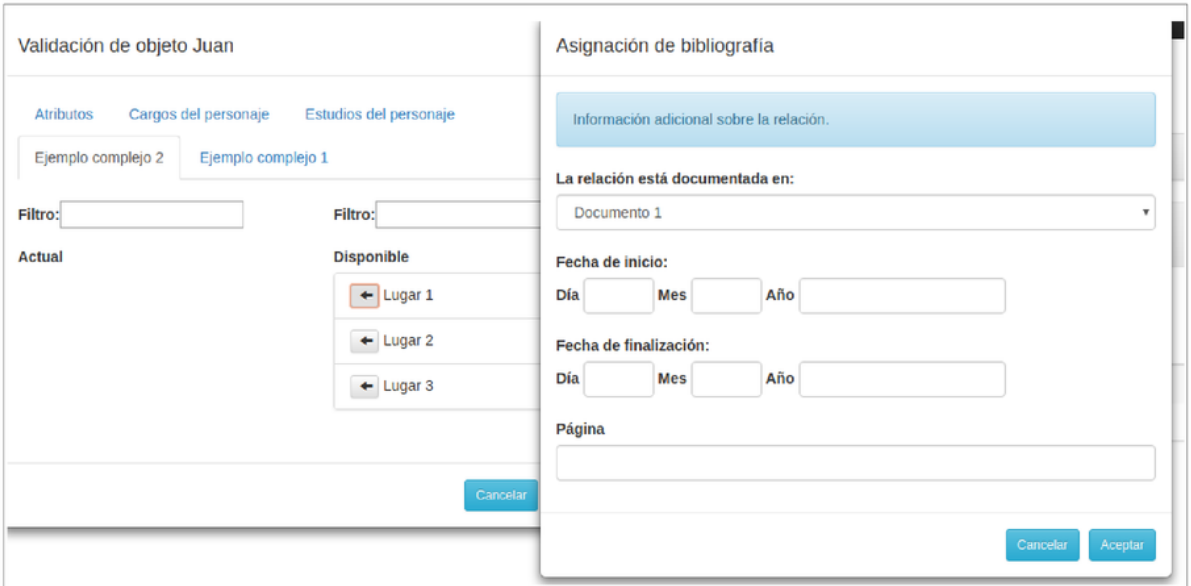

Ilustración 34: Ventana visualizada al añadir una instancia de atributo complejo con información en la relación

Antes de ser añadido "Lugar 1" se ofrece agregar la información complementaria: un objeto de tipo 11 (Documentación), dos fechas y un texto.

Sin embargo, si se hace lo mismo con "Ejemplo complejo 1" no aparecerá dicha ventana.

4.4 Guía de uso

Medievalia está planteado para un uso en cualquier medio con acceso a internet mediante un navegador, por ello tiene un diseño *responsive*. Debido a ello, sigue el modelo de muchas otras páginas y aplicaciones con un menú superior que acaparará gran parte de la navegación.

Según se accede a Medievalia aparecerá la pantalla de inicio de sesión que es común para todos los roles y usuarios. Sin iniciar sesión no se podrá realizar ninguna acción.

La imagen de la derecha muestra su aspecto desde un móvil.

Ilustración 35: Pantalla de inicio de sesión

4.4.1 Guía de administrador

El administrador no tendrá acceso como otros roles al uso final de la aplicación, y solo se centrará en la administración y seguimiento.

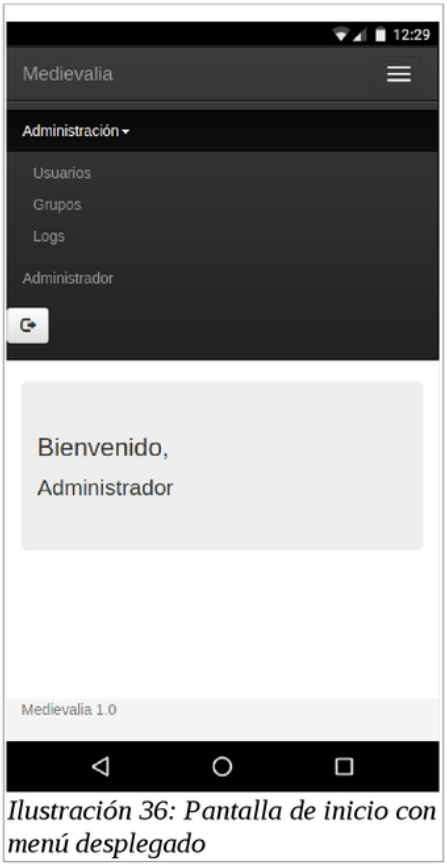

Ilustración 36: Pantalla de inicio con menú desplegado

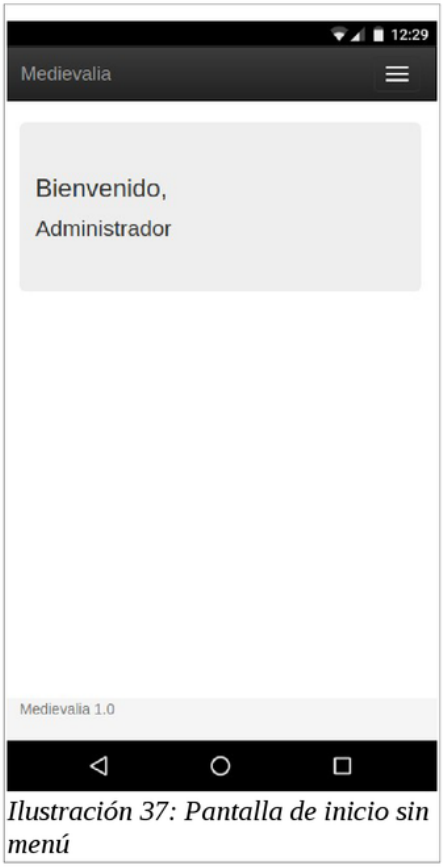

Ilustración 37: Pantalla de inicio sin menú

Todas las opciones accesibles las tiene desde el menú superior . Desde este menú puede acceder a las cuatro pantallas que tiene disponibles. La de grupos es únicamente un listado de los grupos disponibles, sobre los que no cabe acción alguna.

Las otras tres son el perfil, mediante el nombre del usuario, que permite la modificación del mismo, “Usuarios” que lista los usuarios, permite modificarlos y crearlos, y por último la pantalla de “Logs” donde se pueden ver de forma paginada las acciones realizadas por todos los usuarios, con descripciones extendidas sobre su ejecución, si tuvieron éxito o no, fecha y hora.

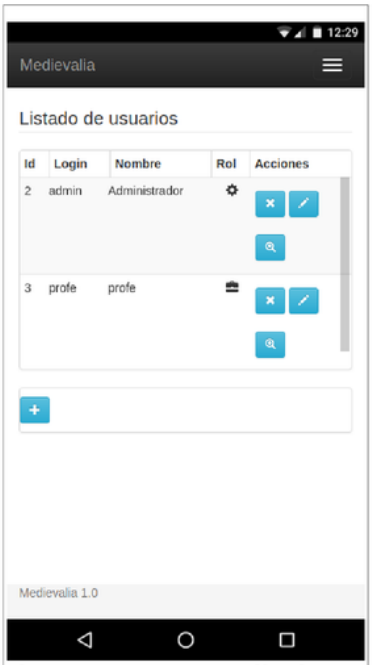

Ilustración 38: Pantalla de lista de usuarios

Ilustración 39: Pantalla de log general

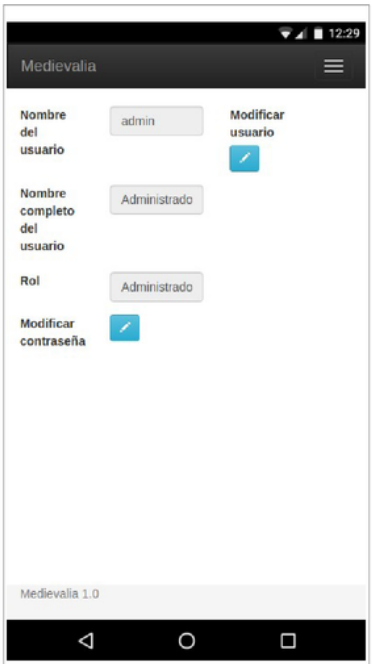

Ilustración 40: Pantalla de perfil de usuario

En la lista de usuarios se puede actuar sobre cada usuario de la lista mediante los botones:

Editar usuario que abrirá una pantalla de edición de perfil de usuario.

Ver actividad del usuario en el log

Eliminar el usuario, aunque esta acción está muy restringida. No se permitirá el borrado mientras el usuario tenga instancias de objeto propias o modificadas, o esté matriculado en algún curso de cualquier forma.

En la parte inferior de la lista de usuarios está el botón para crear usuarios, apareciendo una pantalla de perfil nuevo para editar.

En la pantalla de log general se puede cambiar de página con los botones inferiores numerados o directamente ir a la última y primera páginas.

Por último, en la pantalla de perfil, se ofrece la posibilidad de modificar el nombre de usuario, tanto el de inicio de sesión como el que se visibiliza en la aplicación, así como cambiar la contraseña.

4.4.2 Guía de profesor

La pantalla inicial del profesor contiene un texto y la opción de crear un grupo, además de las opciones del menú superior, que se muestran a continuación.

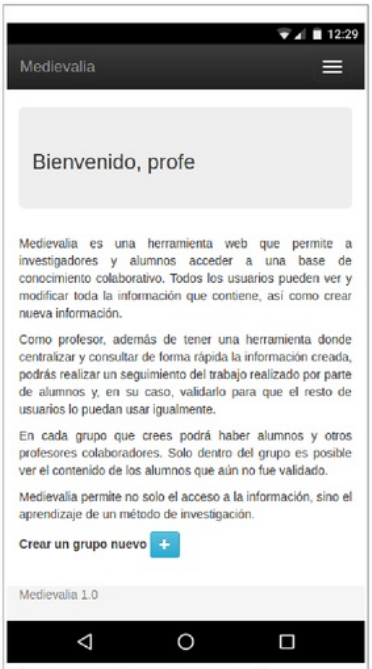

Ilustración 41: Pantalla inicial

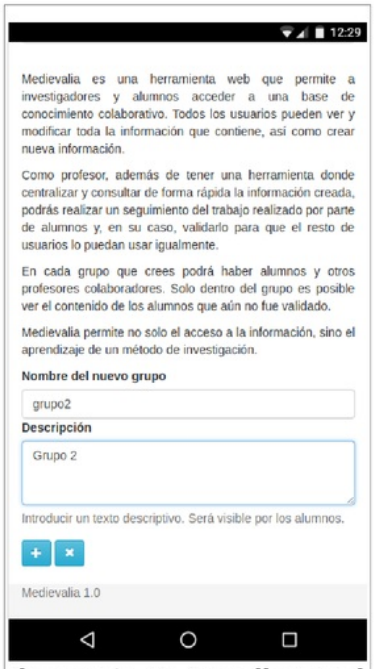

Ilustración 42: Pantalla inicial pulsando en crear grupo

Ilustración 43: Menú superior de profesor

Si se pulsa en crear un grupo nuevo aparecerá un formulario para crearlo. En este se ha de introducir únicamente el nombre y la descripción. Por defecto el profesor que crea un grupo será su director.

Tras cumplimentar el formulario podrá crearse pulsando el botón  o cancelar la operación con el botón  .

En cualquier pantalla que se pulse en el menú superior habrá dos opciones: Grupos y el perfil con el nombre del usuario. En caso de pulsar en grupos se desplegarán los grupos en los que está matriculado el profesor, agrupados en “Dirigidos”, “Como profesor” y “Como alumno” si se da el caso. Si pulsase en el perfil, podrá modificar la información del usuario.

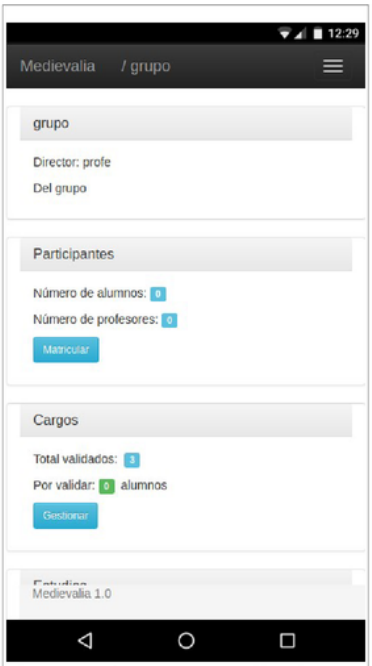

Ilustración 44: Pantalla de gestión de contenido

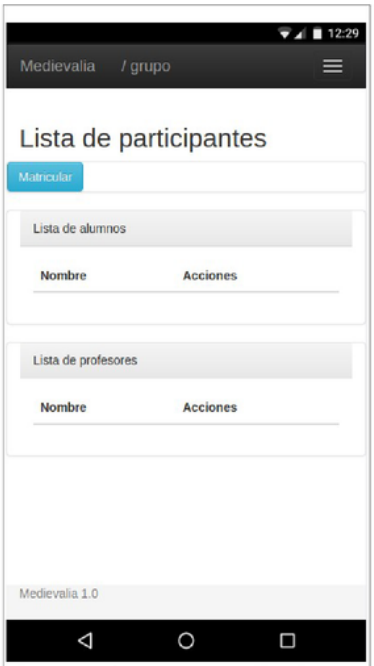

Ilustración 45: Pantalla de matriculación

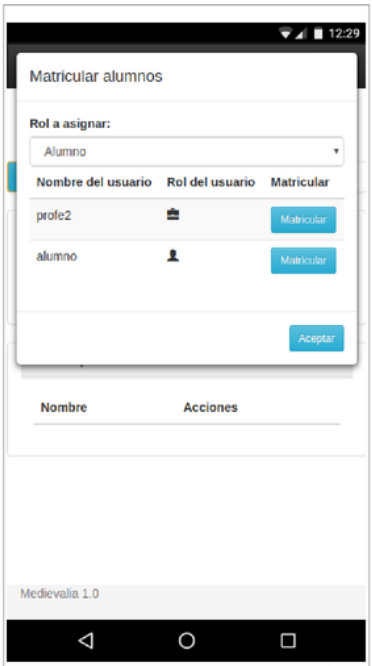

Ilustración 46: Matriculación de usuarios

Una vez seleccione un grupo se abrirá la pantalla de gestión de contenido.En la pantalla de gestión de contenido se muestra una serie de cuadros. El primero es la descripción del grupo. En el segundo aparece información de los usuarios matriculados y el botón de matricular.

Ilustración 47: Cuadro con instancias sin validar

Bajo estos dos cuadros habrá un cuadro por cada tipo de objeto. Se indica la cantidad de instancias validadas que tiene así como cuántas están pendientes de validar y de cuántos alumnos (en las imágenes no hay ninguna, por ello no aparece, aunque sí en la ilustración 47). También hay un botón “Gestionar” que abrirá la pantalla de gestión de objeto.

En la pantalla de “Gestión de objeto” se podrá ver dos secciones colapsadas en la parte superior y bajo ellas dos listas de instancias de objetos del tipo seleccionado. La primera lista será la de instancias por validar del grupo. La segunda lista será la lista completa de instancias, universal para todos los grupos.

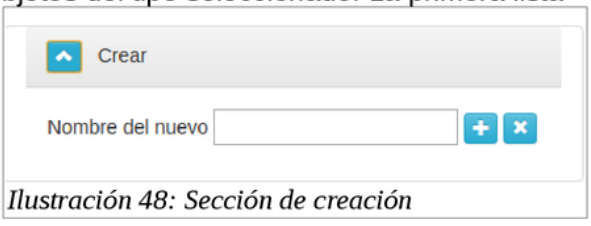

Ilustración 48: Sección de creación

Si se despliega la sección “Crear” aparecerá un

formulario de creación de instancia de objeto, como se puede observar en la ilustración 48. En este solo habrá que introducir el nombre de la nueva instancia que se va a crear.

Ilustración 49: Pantalla de gestión de objeto

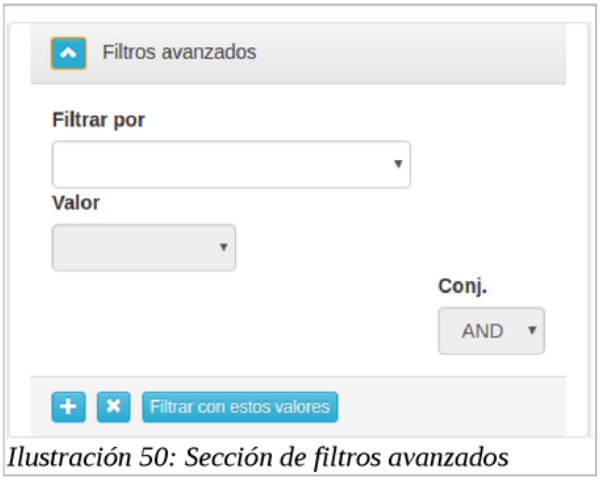

Ilustración 50: Sección de filtros avanzados

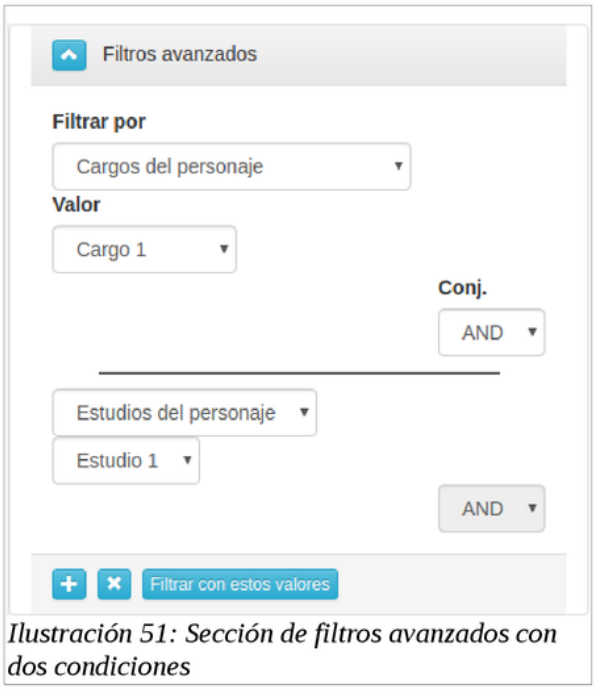

Ilustración 51: Sección de filtros avanzados con dos condiciones

Si se despliega la sección de filtros avanzados se podrá ver un formulario con desplegables en caso de que el tipo de objeto seleccionado tenga atributos complejos, como se puede observar en la ilustración 51. Según se selecciona “Filtrar por” aparecerán los valores posibles en el desplegable “Valor”. Tras elegir un elemento en ambos, se podrá utilizar el filtro avanzado o añadir una condición más con el botón . Además, cuando hay dos o

más condiciones se puede seleccionar también la conjunción entre estas: OR (o) – AND (y).

Se puede eliminar la última condición pulsando el botón  .

El botón 

descargará un archivo pdf con la información detallada de las instancias de la lista completa. Si se ha aplicado un filtro avanzado únicamente aparecerán en dicho informe las instancias seleccionadas, no ocurriendo lo mismo si se usa el filtro por nombre.

En cada lista se podrá ver las instancias que hay.

Ilustración 52: Lista de instancias por validar

Ilustración 53: Lista completa de instancias

En ambas listas se tienen dos botones.

El botón con el icono del ojo permite ver el detalle de la instancia seleccionada. En caso de ser una instancia no validada también da la opción de validarla, y de validar sus atributos complejos.

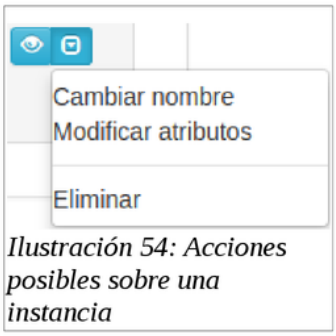

Ilustración 54: Acciones posibles sobre una instancia

El botón de la derecha es desplegable, dando varias opciones según los permisos.

Cambiar el nombre hará que el nombre de la instancia sea editable, y pulsando en el botón  se renombrará, mientras que el botón  deshará los cambios.

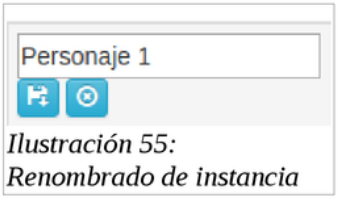

Ilustración 55: Renombrado de instancia

Si se pulsa la opción eliminar aparecerá una ventana para confirmar la acción. Esta acción no se puede deshacer.

Por último, la opción modificar atributos muestra la ventana de edición de la instancia.

Dicha ventana está dividida en pestañas. La primera pestaña, visible en la ilustración 56, está dedicada a los atributos simples de la instancia, para ser modificados según el tipo de atributo que sea: textual, numérico, fecha. Si es un desplegable es debido a que el atributo tomará el valor de un objeto de otro tipo.

Por otro lado, cada una de las siguientes pestañas (ilustración 57) corresponderá a un atributo complejo que pueda tener cada instancia de objeto.

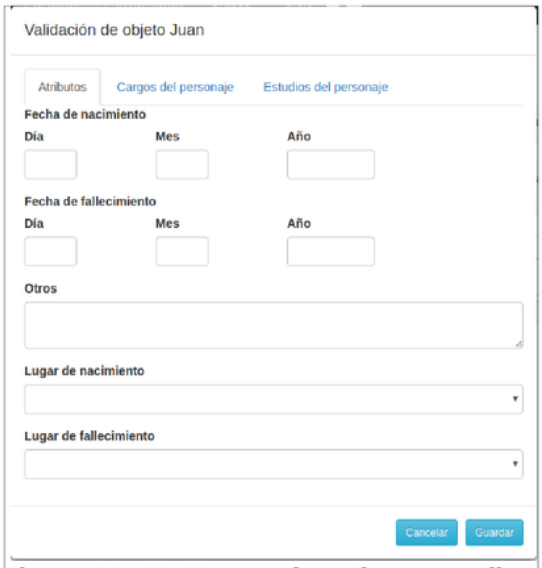

Ilustración 56: Ventana de atributos sencillos de una instancia

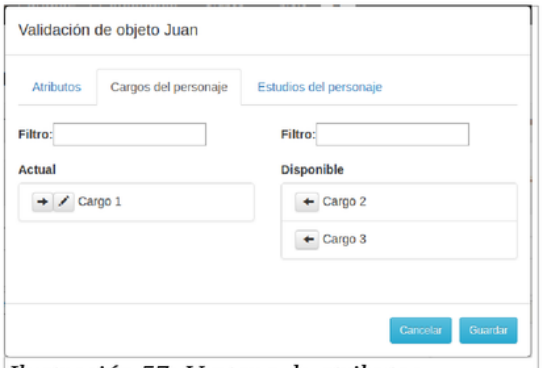

Ilustración 57: Ventana de atributos complejos de una instancia

Estas pestañas tienen dos columnas. La de la izquierda es la lista de instancias de atributo complejo que tiene la instancia que se está editando. La derecha es la lista de las disponibles que podrían añadirse.

Encima de cada lista hay un filtro, para mostrar únicamente las instancias que coincidan con el texto escrito. Se activa simplemente escribiendo en él, o borrando para desactivarlo. Estos son los botones o iconos posibles:

- Elimina una instancia de atributo complejo.
- Edita la relación de la instancia de atributo complejo.
- Añade una instancia de atributo complejo.

El atributo está pendiente de validación.

Permite ver la información de relación.

Cuando un tipo de atributo complejo tiene información de relación, aparecerá la ventana de la derecha (ilustración 58), donde se ofrecen hasta cuatro datos: una instancia de un tipo de objeto, dos fechas y un texto sobre la página (un ejemplo es 55-56, abarcando varias páginas).

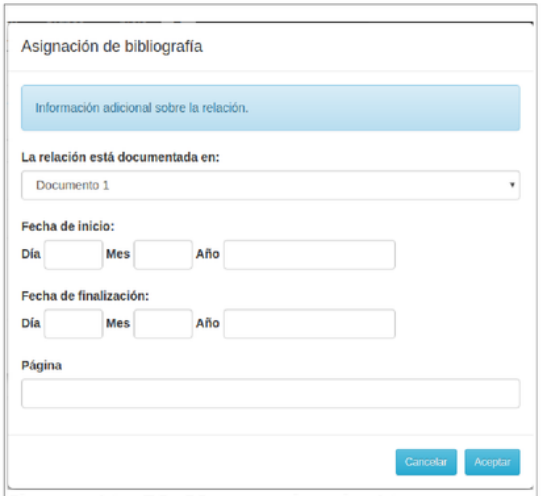

Ilustración 58: Ventana de relación

Sin embargo, si en vez de modificar los atributos se pulsa el botón del ojo de una instancia, no se podrá editar pero sí ver con detalle la instancia de objeto y la información de relación de sus instancias de atributos complejos como se ve en las ilustraciones 59, 60 y 61.

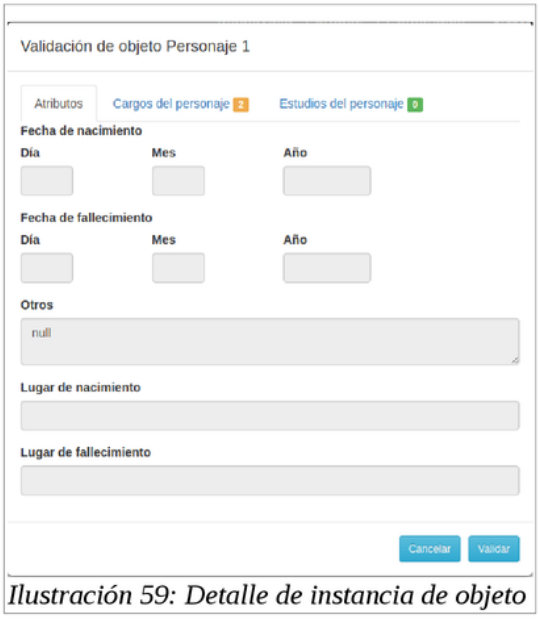

Ilustración 59: Detalle de instancia de objeto

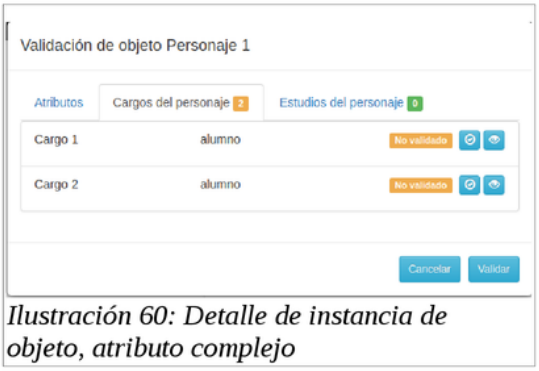

Ilustración 60: Detalle de instancia de objeto, atributo complejo

Ilustración 61: Validación de atributo complejo

Se puede observar en la primera pestaña de la ilustración 59 los atributos sencillos del objeto. Además hay un botón para “Validar” la instancia de objeto en sí, aunque esto no validará las instancias de atributo complejo que contenga.

También se puede apreciar que las pestañas de los diferentes atributos complejos tienen etiquetas numéricas. Estas contienen una etiqueta que informa del número de instancias de atributo complejo que quedan por validar: verde si es 0, amarillo si es mayor.

Si se visualiza una de las otras pestañas se podrá ver la lista de instancias del atributo complejo seleccionado (ilustración 60), indicando si está validado o no, y qué alumno creó la instancia. Hay dos botones a la derecha del todo de cada instancia:

 Permite visualizar la información de la relación de la instancia.

 Abre la ventana de validación del atributo complejo.

En la ventana de validación de atributo complejo (ilustración 61) será posible introducir un texto que podrá ser leído por el alumno, con una etiqueta sobre la instancia a modo de aviso, y ser validado o no. El mensaje se guarda igualmente, sirviendo tanto para sugerir correcciones si no se valida, como para simplemente comentar si se ha validado.

4.4.3 Guía de alumno

La interfaz del alumno es muy similar. La página inicial es idéntica, pero sin la opción de crear un grupo. Igualmente, el menú superior ofrece solo los cursos en los que está matriculado como alumno.

Ilustración 62: Lista de instancias del alumno

La pantalla de “Perfil” es idéntica, y la de “Gestión de Contenido” también, pero sin el botón de “Matricular” por lo que no tiene acceso a la página de matriculación.

La pantalla de “Gestión de objeto” también es idéntica en su mayor parte. Los menús de creación y filtro son iguales, así como la lista completa de instancias.

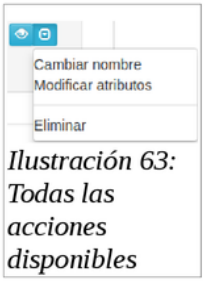

Ilustración 63: Todas las acciones disponibles

Sin embargo la lista de la izquierda corresponde a la lista de instancias creadas o modificadas por el usuario en el grupo actual, por lo que no se ven las instancias de otros alumnos del grupo.

Ilustración 64: Menos acciones disponibles

En ambas listas se puede dar el caso de que sobre cada instancia tenga diferentes permisos a la hora de realizar acciones, dependiendo de si el

objeto es propio del alumno o no, como se observa en las ilustraciones 63 y 64.

No hay diferencia si se accede al detalle de cada instancia, más que la ausencia de los botones de validación.

Ilustración 65: Ventana de detalle de instancia de objeto

A la hora de editar una instancia, la ventana será similar, pero con restricciones. Podrá añadir instancias de atributo complejo nuevas, o eliminar las que haya asignado el mismo alumno y no estén validadas. Pero no podrá modificar los atributos sencillos ni eliminar las instancias de atributo complejo validadas.

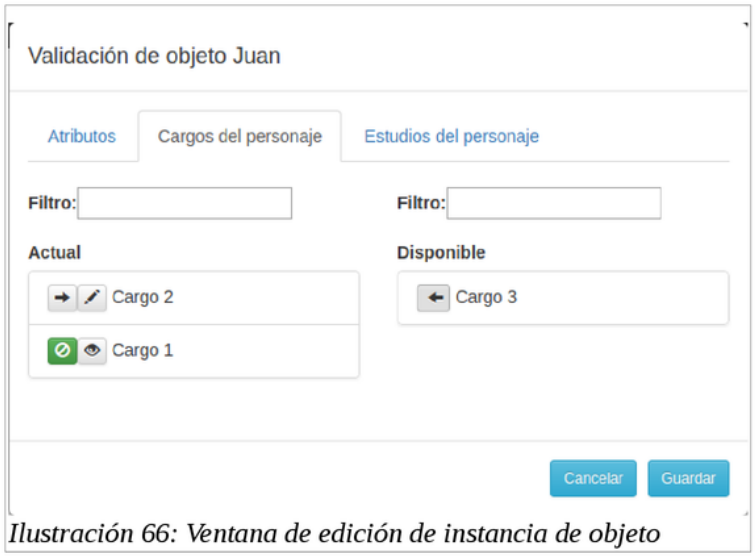

Ilustración 66: Ventana de edición de instancia de objeto

Ilustración 67: Comentarios sin leer

Por ello el alumno se puede encontrar el icono que indica que el atributo está validado y no puede ser eliminado. Igualmente podrá seguir visualizando la información de su relación con el botón .

Por último, si el profesor dejó un comentario de validación de una instancia de objeto, así como de una instancia de atributo complejo, haya sido o no validado, podrá ver una etiqueta de aviso. Si pulsa en ella aparecerá una ventana con el texto del profesor y la opción de marcarlo como leído o no.

4.5 TEMPORALIZACIÓN

El desarrollo de este proyecto ha sido fragmentado en gran medida. Si bien el tiempo total empleado en él, sin incluir la edición de la memoria, es de 278 horas, este se ha extendido desde octubre de 2014 hasta diciembre de 2017 en etapas diferenciadas.

Una primera etapa abarca desde octubre de 2014 hasta marzo de 2015, con el desarrollo inicial. La segunda etapa y tercera etapas se centran en las vacaciones de verano tanto de 2016 y 2017. Y la última etapa, con las últimas modificaciones de calado en el diseño y alcance del proyecto, se sitúan entre septiembre y diciembre de 2017.

El gráfico siguiente ilustra la cantidad de código subido a github (o eliminado) por fecha.

Ilustración 68: Subidas y eliminaciones de código al repositorio

En la primera etapa hay un fuerte pico de subida, que corresponde a la subida inicial. Después de esta no hay subidas hasta el verano de 2016, donde ya hay bastante más trabajo realizado. El pico observado corresponde básicamente a cambios de nombres ya que está compensado por otro de borrado. El verano siguiente se produce una alta producción que luego es compensada en la última etapa al realizar un último cambio profundo del análisis que descarta una parte amplia del trabajo, pero menos significativa y más repetitiva.

En la siguiente tabla se expone un resumen de las tareas realizadas en el tiempo, extraídas de una hoja de cálculo donde se apuntaban todas las actividades realizadas:

Fecha inicio	Fecha fin	Tiempo (h)	Descripción
15/10/14	19/3/15	26:00	Desarrollo de prototipo, servidor Tomcat, configuración de depuración y página inicial.
9/5/16	17/05/16	7:15	Base de datos, autorización y header.
17/5/16	21/8/16	33:15	Gestión de usuarios
25/7/16	5/7/17	18:25	Gestión de grupos
31/7/17	30/8/17	55:10	Gestión de objetos concreta (desechada)
8/9/17	20/11/17	70:55	Nuevo esquema, vista de profesor
4/11/17	29/11/17	19:45	Nuevo esquema, vista de alumno
17/9/17	27/12/17	44:05	Corrección de últimos errores, informes y arreglos de maquetación y diseño gráfico
		3:30	Otros
	Total	278:20	

Tabla 4: Temporalización del proyecto

En cuanto al desarrollo semanal, el trabajo se realizó casi de forma uniforme a lo largo de los días de la semana.

En la tabla resalta una fila en amarillo que está relacionada con el desarrollo de funcionalidad respecto al antiguo esquema en el que la aplicación trabajaba con una base de datos concreta, y no una configurable. Por ello en ese tiempo se desarrollaron gestores concretos de Cargos, Estudios, Personajes, con funcionalidad similar o idéntica. Todo este trabajo no fue descartado por completo, ya que en su mayor parte fue útil para el posterior desarrollo de la gestión de Objetos universal, especialmente la parte de la interfaz web.

Si se analiza el tiempo empleado en diferentes tareas se encontrará el siguiente reparto:

- 3,3% del tiempo empleado en la configuración de equipos, servicios y el entorno de desarrollo.
- 9,27% del tiempo dedicado al diseño y análisis.
- 4,96% del tiempo en desarrollo, diseño y trabajo con la base de datos (no incluye desarrollo de *querys*).
- 42,47% del tiempo en el diseño y programación de las vistas (html, css, Jquery).
- 39,99% del tiempo en programación Java tanto en las capas de modelo, controlador y persistencia, lo cual incluye también algunas consultas de la base de datos.

4.6 CONTRIBUCIONES AL PROYECTO

El autor, al ser el único participante en el proyecto ha llevado la mayor parte de todos los aspectos del proyecto, tanto en la idea inicial, diseño y análisis, como desarrollo, implementación y configuración.

El doctor Óscar Villarroel González, como principal usuario final, ha contribuido con la especificación del alcance y descripción de requisitos. Al plantearse la insuficiencia de una simple aplicación ligada a una base de datos concreta, ideó la posibilidad de añadir a alumnos al proceso y así construir una especie de LMS (Learning Management System) o Sistema de Gestión de Aprendizaje.

El director del proyecto, el doctor Juan Pavón, aceptó el proyecto en un principio como se le planteó con la condición de aumentar la funcionalidad más allá de una simple interfaz web de base de datos, pero posteriormente sugirió la mayor modificación del alcance del proyecto, que fue la universalización del modelo de datos, haciendo que la aplicación fuese configurable para así llegar a otros ámbitos más allá de la investigación en historia, o simplemente la fácil modificación y personalización.

Por otro lado, Magdalena A. Marañon Palma, gracias a sus conocimientos en diseño gráfico e informática, contribuyó a imprimir algunos cambios en el aspecto de la interfaz, pudiendo haber sido muchas más sus contribuciones (sobre todo en cuanto a aspectos estéticos, usabilidad y optimización de la navegación) de no ser por la necesidad del autor de reducir tiempos de desarrollo.

5 Resultados y Conclusiones

5.1 Experiencias de usuario

Medievalia se desplegó en el servidor de producción en www.ghis.ucm.es:8080/medievalia en el mes de diciembre de 2017, disponiéndose cerca de dos semanas para usar la aplicación antes de las vacaciones. Durante ese tiempo y hasta mediados de enero no hubo prácticamente cambios en la aplicación.

5.1.1 Profesorado

En el primer mes de uso únicamente un profesor ha utilizado la aplicación, si bien es cierto que es probable que más profesores se interesen posteriormente, tal y como se describe en el epígrafe siguiente.

La actividad detectada en un mes aproximado de uso (aunque con vacaciones de por medio) aporta los siguientes datos:

- Creación de instancias de objeto: 112.
- Creación de instancias de atributo complejo: 84.
- Modificación de instancias de objeto (puede incluir atributos sencillos y complejos): 570.

En gran parte la actividad detectada fue la inserción inicial de datos, creando instancias de objeto, para que los alumnos tuvieran una base sobre la que trabajar.

Finalmente se solicitó la opinión del profesor sobre el uso de la aplicación y la utilidad para las prácticas que ha propuesto con los alumnos. Se reproduce a continuación:

> Óscar Villarroel González
>
> Experiencia de usuario Medievalia
>
> Categoría: profesor
>
> A lo largo de estos primeros días de trabajo con Medievalia me he conectado más de una vez al día (en días de diario). A nivel de usuario estándar (dado que como profesor e investigador el uso de la aplicación para la investigación de base tiene también un gran potencial) el trabajo en un primer momento se hace complicado, al tener que completar diversas instancias, pero una vez que se coge el hábito y el orden de trabajo,

es sencillo.

Es potente además en cuanto a sus posibilidades en la investigación histórica, dado que que se tienen en cuenta muchos parámetros que luego nos permiten análisis muy diversos.

Como profesor he podido experimentar también con la aplicación tutorizando a un grupo de cuatro alumnos. Estos se iniciaban en la investigación y no tenían experiencia de trabajo con bases de datos. Al estar inmersos en unas prácticas extracurriculares el trabajo con la aplicación era la base para ellos. En su formación supone experimentar con la metodología de identificar, siglar, y describir los datos de una forma homogénea, sabiendo discriminar los datos relevantes en cada ocasión. Posteriormente llegará el momento de que aprecien su utilizar en la fase de análisis y redacción de resultados.

El trabajo con los alumnos fue rápido y sencillo en cuanto al manejo de la aplicación y rápidamente se hicieron con la tónica de trabajo. La supervisión como profesor ha permitido analizar su evolución y, sobre todo, cómo asimilaban el método de trabajo con diversas fuentes de información. De esta forma se ha podido comprobar cómo aprendían del error y no se repetían en lo sucesivo. La validación o no validación de las acciones de los alumnos (así como el poder indicarles el error cometido o los aspectos a completar) permitía una retroalimentación constante y en tiempo real del alumno. La virtualidad además nos ha permitido trabajar en ámbitos distintos. Sí es cierto que en alguna ocasión la necesidad de validar ciertos atributos supone una necesidad de atención permanente por parte del profesor (de forma que los alumnos puedan seguir trabajando), pero sin duda permite un control exhaustivo de cómo lo están haciendo. Es en esa fase de validación donde se han podido detectar los errores cometidos, de todo tipo: identificaciones erróneas o generalizaciones, datos incompletos, análisis erróneo de fuentes, asignación errónea de atributos… De modo que para la formación tiene un valor fundamental. Además, una vez que los alumnos se desenvuelvan de forma perfecta el administrador siempre puede considerar el subirles de categoría para no estar sujetos ya a validación.

La deslocalización ha sido una posibilidad también empleada: el no ser necesario estar junto al alumno para estar trabajando ha sido una herramienta ampliamente utilizada. Incluso en las primeras semanas de trabajo con los alumnos, por circunstancias sobrevenidas, ha habido que trabajar de esta forma, demostrándose como una posibilidad real y que no ralentiza el trabajo. Si la atención del profesor se mantiene la

tarea se desarrolla con normalidad.

Es muy valorable también la capacidad de acceso multiplataforma. Se ha trabajo principalmente en entorno Windows, pero en ocasiones también con Android y, excepcionalmente, con Mac. Esa posibilidad, además, ha permitido que cada alumno trabaje con su propio equipo, lo que facilita además la movilidad.

En mi opinión la aplicación permite mostrar a los alumnos la metodología de trabajo con bases de datos en la investigación histórica de una forma cómoda. Además, al trabajarse en un entorno colaborativo (lo que no es habitual en ese campo) se logra una mayor amplitud en los resultados posibles al ampliarse la base documental y los puntos de vista. El equipo de trabajo, así, ve la posibilidad de verse beneficiado del trabajo de los compañeros, sabiendo que ellos, a su vez, pueden nutrirse del suyo.

Por último, la aplicación se muestra lo suficientemente versátil como para ser utilizadas en otras circunstancias en los que el trabajo de investigación histórica y con alumnos estén presentes. En la medida de lo posible se utilizará para prácticas en grupos de grado (de los cursos superiores), puesto que supone una útil herramienta para que los alumnos se acostumbren a una metodología de trabajo exhaustiva y completa.

5.1.2 Alumnado

Debido a que en ciertas asignaturas de grado no era posible añadir prácticas con la aplicación sin antelación, finalmente se contó con cuatro alumnos del Máster de Estudios Medievales para el uso de la primera versión, de forma que fuese parte de las propias prácticas del máster, y no como actividad voluntaria. En el segundo cuatrimestre del curso 2017-2018 será posible añadir a alumnos estudiantes de grado, siendo estos mucho más numerosos.

Estos alumnos, tras más de un mes desde el primer uso (aunque con vacaciones de navidad incluidas) han utilizado la aplicación con cierta intensidad, tal y como ilustran los siguientes datos obtenidos de la base de datos:

- Creación de instancias de objeto por alumno: 64, 95, 94 y 39.
- Creación de instancias de atributo complejo por alumno: 73, 176, 112 y 91.
- Modificación de instancias de objeto (puede incluir atributos sencillos y complejos): 348, 733, 489 y 266.

Adicionalmente a la actividad registrada por la aplicación, se realizó un formulario preguntando a los alumnos sobre el uso de la aplicación. Las preguntas fueron sobre:

- Frecuencia de uso de la aplicación, relativa a las horas de prácticas:

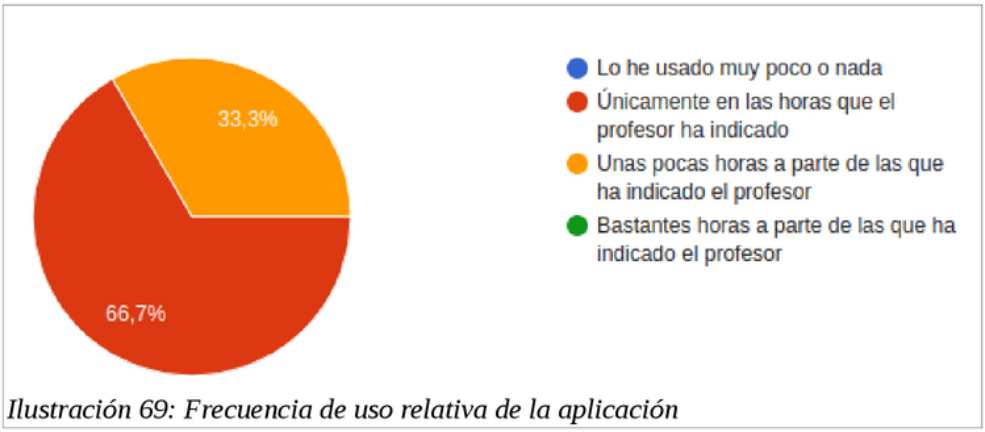

Ilustración 69: Frecuencia de uso relativa de la aplicación

- El lugar de conexión. Todos se conectaron desde el aula de prácticas, un 66% también desde casa y un 33% desde la biblioteca.
- El medio utilizado. Todos usaron un portátil. Ninguno utilizó además otro medio.
- En una escala de 1 a 5, donde 1 es Muy difícil y 5 muy fácil, se preguntó por la facilidad de aprender a usar la aplicación.

Ilustración 70: Valoración de la facilidad de aprendizaje de la aplicación

- La comodidad en el uso de la aplicación, donde 1 es muy incómoda y 5 muy cómoda.

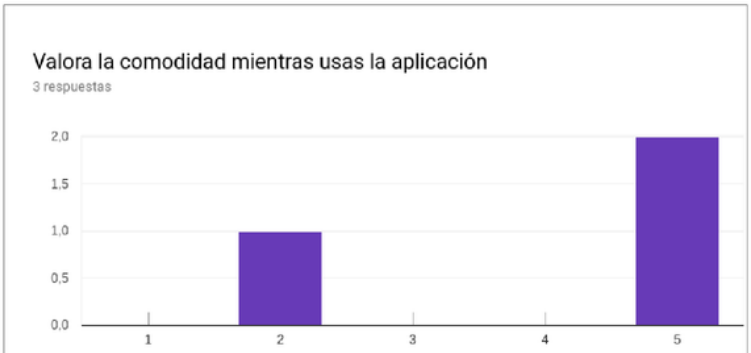

Ilustración 71: Valoración de la comodidad de uso

- Respecto a lo que ayudó la aplicación para la comprensión del método de trabajo que quiere inculcar el profesor, todos respondieron “Sí, ahora comprendo muy bien cómo se ha de trabajar”.
- La utilidad del proceso de validación y los mensajes, en una escala de 1 a 5, donde 1 es incómodo y nada útil y 5 es cómodo y útil.

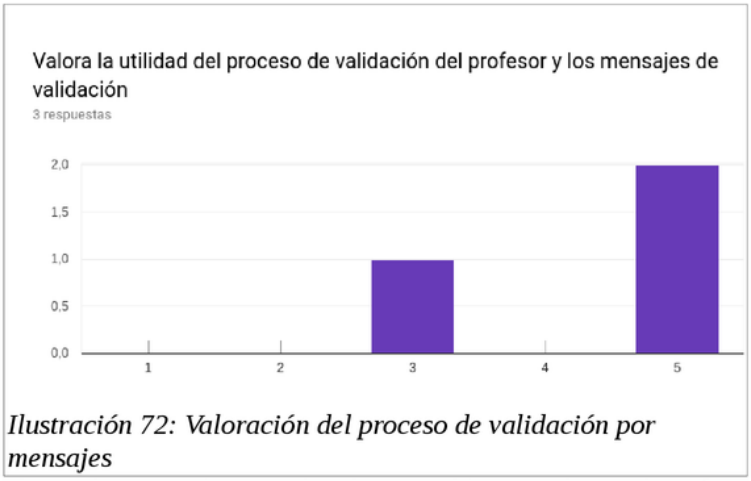

Ilustración 72: Valoración del proceso de validación por mensajes

- La utilidad de poder acceder a todo el contenido que tiene el profesor y otros grupos, en una escala del 1 al 5, donde 1 es nada útil y 5 muy útil, el 100% respondió que le resulta muy útil.

- Usar la aplicación en otras asignaturas.

Ilustración 73: Aplicación a otras asignaturas

Lo que se extrae de esta encuesta, sabiendo que es pequeña la muestra, es que los alumnos en general valoran positivamente el uso y los aportes a su formación de la aplicación, pero que con todo puede mejorar en la portabilidad a otras asignaturas y la comodidad de uso. También desvela que el uso con dispositivos móviles es nulo de momento entre los alumnos.

5.2 Conclusiones

A nivel funcional Medievalia ha alcanzado todos los objetivos que se marcó desde septiembre y octubre de 2017, que fue cuando se concretaron en su versión final. Si bien es cierto que una vez finalizado el desarrollo o cerca de su final se detectaron nuevas necesidades para el proyecto, algunas sencillas se incorporaron al mismo, pero otras se dejaron para posteriores desarrollos. Esto se detalla en versiones futuras, en el epígrafe 4.4.

Tras la puesta en funcionamiento con usuarios reales se ha mostrado como una herramienta útil, y que a corto plazo tendrá un uso continuado, por lo que puede preverse que no caerá en el desuso. En definitiva una aplicación que, tal y como fue ideada, cumple una funcionalidad y es útil para sus usuarios.

Quedará por ver en el futuro si su aceptación aumenta, tal y como sugieren los profesores, o se queda reducida a un número pequeño de usuarios. De igual forma, habrá que dilucidar si

otras áreas, probablemente con la sociología en primer lugar, se interesan en aplicarla a sus estudios e investigaciones.

5.3 MÉTRICAS

Hay diferentes formas de medir el tamaño de la aplicación web. Algunas de estas formas se detallan a continuación:

- 51 Unidades funcionales: pueden ser perfectamente representadas por la cantidad de controladores que existen. Están divididas en 34 asíncronas y 17 síncronas. Pero hay una gran divergencia de complejidad y tamaño entre ellas.
- Líneas de código: aunque este método puede ser un tanto subjetivo, el llegar a cierto tamaño evidencia la magnitud del proyecto. Para ello se pueden usar diferentes herramientas, que entre otras políticas, ignoran líneas en blanco y comentarios. En este caso se utilizó “Lines of Code Wichtel” (http://www.andreas-berl.de/linesofcodewichtel/en/download.html). Los resultados se muestran a continuación divididos por tipos y carpetas del proyecto:
  - Total líneas procesadas 37884, con 25606 líneas de código, 11480en blanco y 798 de comentarios.
  - 7052 líneas de código Java en los paquetes propios de la aplicación.
  - 4358 de .jsp para la construcción de la interfaz web.
  - 206 líneas .xml en ficheros de configuración.
  - 3359 líneas de .js propias del autor (JavaScript, Jquery, AJAX).
  - 102 líneas de .css propias del autor.
  - Bootstrap está incluido en el proyecto, conteniendo 9374 líneas de css y 1248 de js no desarrolladas por el autor.
- La base de datos necesaria para la instalación se instala mediante la importación de un archivo .sql que incluye:
  - 20 creaciones de tablas.

- 16 alteraciones de tablas para añadir claves ajenas.
- 163 inserciones de registros en tablas con configuración básica de la aplicación y del esquema por defecto.

5.4 VERSIONES FUTURAS

Respecto al diseño inicial se alcanzaron todos los objetivos definidos. Sin embargo durante el desarrollo y especialmente al final del mismo cuando ya había usuarios finales haciendo uso de la aplicación, se propusieron algunas nuevas funcionalidades que no llegaron a incluirse en el proyecto. Otras también fueron ideadas por el propio autor, pero tampoco se incluyeron. Algunas, las más significativas, se describen a continuación:

- Guardar información sobre las actividades más usuales de cada usuario y con ello ofrecer en el menú superior una lista de enlaces a dichas actividades. Esto conllevaría un cambio en la base de datos, así como en la clase HtmlManager de la capa de modelo y crear una nueva clase asociada en la de persistencia. Igualmente, habría que diseñar una política adecuada a la hora de elaborar la lista a ofrecer, para mayor utilidad.
- Los filtros avanzados para la lista de instancias de objeto tienen un gran alcance en cuanto a mejoras.
  - Añadir búsquedas por atributos sencillos.
  - Añadir filtro según la información de relación, asociado al filtro ya existente.
  - Para ciertos campos de atributos sencillos o las fechas de relación, añadir comparadores como mayor o menor que.
- Un requisito relativamente sencillo pero con mucha utilidad, pero también impacto, sería la posibilidad de que los alumnos puedan cambiar la información de relación. Esto necesitaría un análisis pausado sobre las consecuencias en la autoría de la instancia de atributo complejo, una capa extra de validación que iría más allá de la de instancia de objeto e instancia de atributo complejo.
- Un módulo completo de funcionalidades nuevas para el administrador orientadas a la

modificación del modelo de datos de forma sencilla e intuitiva, en vez de configuración manual a través de adición de registros a la base de datos.

- La clase FechaEspecial podría incluir tratamiento de incertidumbre. Cuanto más antiguo es el periodo histórico menos certeras pueden ser las fechas, incluso contando con diferentes formas de calcularlas, como sucede en historia antigua, o que surja a partir de métodos físicos como pruebas de carbono. Este tratamiento podría incluir rangos de fechas.
- Una funcionalidad con poco impacto en cuanto a controladores y persistencia, pero con mayor complicación en las vistas para ser realmente útil y con un acabado óptimo, sería la adición de imágenes a instancias de objeto. Estas, en el ámbito de investigación histórico, serían muy útiles a la hora de almacenar los documentos con la fuente primaria de los datos.

6 BIBLIOGRAFÍA

- Spring
  - Just Spring, Madhusudhan Konda. Sebastopol, CA : O'Reilly, c2011
  - Introducing Spring Framework, Felipe Gutierrez. Berkeley, CA : Apress : Imprint: Apress, 2014
  - Tutorial para el prototipo:
    - https://www.uv.es/grimo/teaching/SpringMVCv3PasoAPaso/part1.html
- Bootstrap 3.3
  - https://getbootstrap.com/docs/3.3/
- Jquery y AJAX:
  - http://api.jquery.com/
- Encuesta de valoración de la aplicación:
  - https://docs.google.com/forms/d/e/1FAIpQLScSvPJUXEy5UZ_pi4gfRvGSIhBqPNsLLaeiWHfa8EWBqqjCjw/viewform?usp=sf_link

7 ANEXOS

7.1 ANEXO I. TABLAS DE LA BASE DE DATOS

En adelante se muestran los detalles de todas las tablas. PK: identificador. FK: clave ajena. AI: autoincremento. UN: único.

- ObjetoDom. Describe los tipos de objetos que habrá.

ObjetoDom		
idObjeto	int	AI. PK. Identificador
nombreObjeto	varchar(100)	Nombre con el que se mostrará el tipo de objeto

- AtributoSencilloObjeto. Describe los atributos sencillos que tendrá cada tipo. Cada instancia indicará un atributo que tendrá dicho objeto que solo adquirirá un valor. El idAtributo no es autoincremental, sino que es un ordinal indicando el número de atributo sencillo de cada objeto. Cada tipo de objeto tendrá (o no) atributos sencillos numerados desde 1 hasta n.

AtributoSencilloObjeto		
idAtributo	int	PK. Identificador
idObjeto	int	PK. FK a ObjetoDom indicando qué tipo tiene un atributo
tipo	int	FK a Tipo. Indica uno de los 6 tipos soportados
nombreAtributo	varchar(50)	Nombre que se mostrará como atributo del tipo de objeto en sus instancias. Por ejemplo “Fecha de nacimiento”.
subtipo	int	En caso de ser tipo Objeto, indicará qué tipo de objeto es.

- AtributoComplejoObjeto. Describe los atributos complejos que tendrá cada tipo. Cada instancia indicará un atributo que tendrá dicho objeto que podrá adquirir múltiples valores.

AtributoComplejoObjeto		
idObjetoPadre	int	PK. FK a ObjetoDom indicando qué tipo tiene un atributo.
idObjetoHijo	int	PK. FK a ObjetoDom indicando de qué tipo tiene el atributo.
nombreAtributo	varchar(100)	Nombre que se mostrará como atributo del tipo de objeto en sus instancias. Por ejemplo “Implicados”
idObjetoRelacion	int	Puede ser nulo o indicar qué tipo de objeto se asocia cuando se hace la relación entre instancias padre-hijo
conFecha	int	Indica si el atributo asocia fechas de inicio y fin a la relación (1) o no (0) cuando se hace la relación entre instancias padre-hijo
conPaginaDoc	int	Indica si el atributo asocia una cadena de caracteres que aclare el objeto relación (1) o no (0). En la aplicación histórica es una o varias páginas o secciones: “12-22”

- InstanciaObjeto. Cada objeto que creen los usuarios será reflejado en esta tabla, donde se indica el tipo que tiene así como otros valores del mismo. En este caso el identificador tampoco es autoincremental, teniendo, en caso de existir, cada tipo de objeto una instancia 1, 2, etc. El nombre no es único dado que se controla por la aplicación que cada tipo no tenga instancias con el mismo nombre, pero no se impide que diferentes tipos de instancias sí se llamen igual.

InstanciaObjeto		
idInstancia	int	PK. Identificador
idObjeto	int	PK. FK a ObjetoDom indicando qué tipo de objeto es
nombreInstancia	varchar(100)	Nombre que da el usuario
validado	int	Si está validado (1) o no. Los profesores crean instancias validadas y validan las de los alumnos que se crean no validadas
textoValidacion	varchar(255)	El texto orientativo que deja un usuario profesor al validar o no una instancia que creó un alumno
idGrupo	int	FK a Groups. Indica la pertenencia a un grupo.
creador	int	FK a Users. Indica el usuario que creó la instancia.
textoLeido	int	Indica si el alumno marcó como leído textoValidación (1) o no (0).

- Tipos. Tiene 6 tipos que la aplicación admite. No sería suficiente con añadir registros

para que la aplicación soporte más. Esta parte necesitaría ser programada. Los tipos existentes son: 1 date, 2 double, 3 int, 4 varchar(255), 5 longtext(65535), 6 objeto.

Tipos		
idTipo	int	PK. Identificador
descripcion	varchar(10)	Nombre técnico: Date, String, Text, Double...
descLarga	varchar(50)	Descripción del tipo

- InstanciaAtributoInt. Cuando una instancia tiene un valor asignado a un atributo sencillo tendrá un registro en esta tabla indicando a qué objeto, de qué tipo y de qué atributo sencillo se trata. La PK impide más de un atributo sencillo de un tipo a una misma instancia.

InstanciaAtributoInt		
idInstancia	int	PK. FK a InstanciaObjeto. Indica qué instancia de objeto tiene el atributo
idObjeto	int	PK. FK a InstanciaObjeto. La misma clave ajena, indicando qué tipo de objeto es la instancia
idAtributoSencillo	int	PK. FK a AtributoSencilloObjeto. Indica el atributo sencillo que corresponde a la instancia.
valor	int	Valor numérico entero del atributo

- InstanciaAtributoString. Al igual que la tabla anterior, asigna un valor a un atributo sencillo pero textual hasta 255 caracteres.

InstanciaAtributoString		
idInstancia	int	PK. FK a InstanciaObjeto. Indica qué instancia de objeto tiene el atributo
idObjeto	int	PK. FK a InstanciaObjeto. La misma clave ajena, indicando qué tipo de objeto es la instancia
idAtributoSencillo	int	PK. FK a AtributoSencilloObjeto. Indica el atributo sencillo que corresponde a la instancia.
valor	varchar(255)	Valor textual del atributo

- InstanciaAtributoText. Como las otras tablas anteriores, pero con un valor textual de hasta 65535 caracteres.

InstanciaAtributoText		
idInstancia	int	PK. FK a InstanciaObjeto. Indica qué instancia de objeto tiene el atributo
idObjeto	int	PK. FK a InstanciaObjeto. La misma clave ajena, indicando qué tipo de objeto es la instancia
idAtributoSencillo	int	PK. FK a AtributoSencilloObjeto. Indica el atributo sencillo que corresponde a la instancia.
valor	longtext	Valor textual del atributo

- InstanciaAtributoDouble. Mismo funcionamiento que los anteriores, pero con un valor numérico real.

InstanciaAtributoDouble		
idInstancia	int	PK. FK a InstanciaObjeto. Indica qué instancia de objeto tiene el atributo
idObjeto	int	PK. FK a InstanciaObjeto. La misma clave ajena, indicando qué tipo de objeto es la instancia
idAtributoSencillo	int	PK. FK a AtributoSencilloObjeto. Indica el atributo sencillo que corresponde a la instancia.
valor	double	Valor numérico real del atributo

- InstanciaAtributoObjeto. Esta última tabla tiene una diferencia con las anteriores y es que referencia a un registro de la tabla InstanciaObjeto.

InstanciaAtributoObjeto		
idInstancia	int	PK. FK a InstanciaObjeto. Indica qué instancia de objeto tiene el atributo
idObjeto	int	PK. FK a InstanciaObjeto. La misma clave ajena, indicando qué tipo de objeto es la instancia
idAtributoSencillo	int	PK. FK a AtributoSencilloObjeto. Indica el atributo sencillo que corresponde a la instancia
idObjetoHijo	int	FK a InstanciaObjeto con el tipo de objeto
idInstanciaHijo	int	FK a InstanciaObjeto con la instancia referenciada

- Role. Esta tabla recoge los diferentes roles que existen en la aplicación. Podría modificarse para añadir o quitar roles, teniendo efecto inmediato. Los roles por defecto son otro (0) administrador (1), profesor (2), alumno (3) e inactivo (4). El rol 0 se usa

cuando no hay usuario, como en los intentos de sesión fallidos. El usuario 4 se usa como alternativa de borrado de usuarios.

Role		
idRol	int	PK. Identificador
nombreRol	varchar(20)	Nombre del rol con el que se visualizará en la interfaz

- Action. Describe todas las acciones que se realizan en la aplicación. Son un reflejo de las funciones programadas.

Action		
idAction	int	PK. Identificador
action_name	varchar(50)	Descripción de la acción

- Authorization. Son la relación entre acción y rol. Cuando existe un registro con una acción y un rol significa que los usuarios con el rol indicado podrán realizar la acción indicada. Si no existe, no habrá permiso y no se realizará la operación.

Authorization		
idAction	int	PK. FK a Action
idRol	int	PK. FK a Role

- Users. Contiene la lista de usuarios, tanto los activos (roles 1 a 3) como los inactivos (rol 4) para preservar así histórico y los objetos creados por los usuarios inactivados. Hay un usuario especial con id 0, que se usa cuando una acción se lleva a cabo cuando no hay usuario identificado, como al acceder a la pantalla de bienvenida antes de iniciar sesión.

Users		
user_id	int	PK. AI. Identificador
user_name	varchar(20)	Nombre del usuario e identificador de inicio de sesión
user_long_name	varchar(50)	Nombre y apellidos del usuario. Será la visualización típica en la aplicación
user_pass	varbinary(250)	Campo donde se guarda la contraseña del usuario. Está encriptada mediante el algoritmo AES
user_role	int	FK a Role. Indica el rol del usuario

- Log. Esta tabla lleva un registro de todas las acciones que se han realizado

Log		
idLog	int	PK. Identificador
idUser	int	FK a Users. Indica el usuario que realizó la acción
time	datetime	Indica la fecha y hora en que se realizó
idAction	int	FK a Actions. Especifica la acción realizada
descripcion	varchar(1000)	Describe la acción que se realizó, pudiendo especificar detalles como casos concretos de la acción o los parámetros
success	int	Indica si la acción tuvo éxito (1) o no (0)

- Groups. Contiene la lista de grupos existentes.

Groups		
idGroup	int	PK. Identificador
name	varchar(100)	Nombre del grupo que será visualizado
director	int	FK a users
descripcion	varchar(250)	Descripción del grupo indicada por el profesor que crea el grupo y es visualizada por los usuarios que entren

- Students. La tabla indica los usuarios matriculados como alumnos en un grupo. Un usuario con rol profesor puede ser matriculado como alumno en un grupo que no dirige.

Students		
idStudent	int	PK. FK a Users
idGroups	int	PK. FK a Groups

- Teachers. Indica la lista de usuarios matriculados como profesor en el grupo. Solo usuarios con rol profesor pueden estar así matriculados.

Teachers		
idTeacher	int	PK. FK a Users
idGroups	int	PK. FK a Groups

7.2 Anexo II. Casos de Uso

CU-Sin-Sesion-01	Visita portal
Objetivo en contexto	Obtener la Pantalla de inicio de sesión
Precondiciones	Tomcat y base de datos funcionando, aplicación cargada.
Postcond. si éxito	Carga de la pantalla
Postcond. si fallo	Error 404 o servidor no encontrado
Actores	Sin usuario en sesión
Secuencia normal	1. El usuario introduce la url, puerto 8080 seguido de /medievalia

CU-Sin-Sesion-02	Inicio de sesión
Objetivo en contexto	Autenticar el usuario y guardarlo en sesión
Precondiciones	Usuario y contraseña correctos
Postcond. si éxito	Carga Pantalla de inicio
Postcond. si fallo	Vuelta a Pantalla de inicio de sesión
Actores	Cualquier usuario del sistema
Secuencia normal	1. El usuario introduce usuario y contraseña 2. Pulsa el botón Iniciar Sesión

CU-Header-01	Seleccionar grupo
Objetivo en contexto	Seleccionar un grupo en el que el usuario esté matriculado
Precondiciones	Sesión iniciada
Postcond. si éxito	Carga pantalla de Gestión de Contenido
Postcond. si fallo	Mensaje de error de parámetros
Actores	Usuarios con rol profesor o alumno
Secuencia normal	1. Pulsar en el menú superior, en cualquiera de los grupos del desplegable grupos.

CU-Header-02	Finalizar sesión
Objetivo en contexto	Eliminar el usuario almacenado en sesión
Precondiciones	Sesión iniciada
Postcond. si éxito	Carga Pantalla de inicio de sesión
Postcond. si fallo	Mensaje de error de parámetros
Actores	Cualquier usuario
Secuencia normal	1. Pulsar en el menú superior en el botón blanco con icono de salida

CU-Header-03	Perfil
Objetivo en contexto	Ver la pantalla de Perfil
Precondiciones	Sesión iniciada
Postcond. si éxito	Carga Pantalla de Perfil
Postcond. si fallo	Mensaje de error de parámetros
Actores	Cualquier usuario
Secuencia normal	1. Pulsar en el menú superior en el enlace con el nombre completo del usuario

CU-Header-04	Seleccionar grupo
Objetivo en contexto	Seleccionar un grupo en el que el usuario esté matriculado
Precondiciones	Sesión iniciada
Postcond. si éxito	Carga pantalla de Gestión de Contenido
Postcond. si fallo	Mensaje de error de parámetros
Actores	Usuarios con rol profesor o alumno
Secuencia normal	1. Pulsar en el menú superior, en cualquiera de los grupos del desplegable grupos

CU-Header-05	Gestión de usuarios
Objetivo en contexto	Cargar la pantalla de gestión de usuarios
Precondiciones	Sesión iniciada
Postcond. si éxito	Carga Pantalla de Gestión de Usuarios
Postcond. si fallo	Mensaje de error de autorización
Actores	Usuarios con rol administrador
Secuencia normal	1. Pulsar en el menú superior el enlace Usuarios en Administración

CU-Header-06	Lista de Grupos
Objetivo en contexto	Cargar la Pantalla de Lista de Grupos
Precondiciones	Sesión iniciada
Postcond. si éxito	Carga Pantalla de Lista de Grupos
Postcond. si fallo	Mensaje de error de autorización
Actores	Usuarios con rol administrador
Secuencia normal	1. Pulsar en el menú superior el enlace Grupos en Administración

CU-Header-07	Log General
Objetivo en contexto	Cargar la Pantalla de Log General
Precondiciones	Sesión iniciada
Postcond. si éxito	Carga Pantalla de Log General
Postcond. si fallo	Mensaje de error de autorización
Actores	Usuarios con rol administrador
Secuencia normal	1. Pulsar en el menú superior el enlace Logs en Administración

CU-Inicio-01	Seleccionar grupo
Objetivo en contexto	Añadir grupo actual a sesión
Precondiciones	Sesión iniciada
Postcond. si éxito	Carga Pantalla de Gestión de contenido
Postcond. si fallo	Mensajes de error
Actores	Usuarios con rol alumno o profesor
Secuencia normal	1. Pulsar en el menú superior el nombre del grupo en el desplegable Grupos, y los submenús Dirigidos, Como profesor o Como alumno

CU-Inicio-02	Crear grupo
Objetivo en contexto	Crear un grupo
Precondiciones	Sesión iniciada
Postcond. si éxito	Mensaje de éxito
Postcond. si fallo	Mensajes de error
Actores	Usuarios con rol profesor
Secuencia normal	1. Pulsar en el botón “+” y el texto “Crear grupo”

CU-GestCon-01	Seleccionar un tipo de objeto
Objetivo en contexto	Cargar la página de gestión de objeto
Precondiciones	1. Sesión iniciada 2. Grupo seleccionado
Postcond. si éxito	Carga de la página
Postcond. si fallo	Pantalla de error
Actores	Usuarios con rol profesor
Secuencia normal	1. Pulsar el botón “Gestionar” del objeto deseado

CU-GestCon-02	Gestión de matriculación
Objetivo en contexto	Cargar la página de Matricular usuarios
Precondiciones	1. Sesión iniciada 2. Grupo seleccionado 3. Ser director o profesor del grupo
Postcond. si éxito	Carga de la página
Postcond. si fallo	Pantalla de error
Actores	Usuarios con rol profesor
Secuencia normal	1. Pulsar el botón “Matricular”

CU-Matricula-01	Listado de usuarios disponibles
Objetivo en contexto	Matricular un usuario
Precondiciones	1. Sesión iniciada 2. Grupo seleccionado 3. Ser director o profesor en el grupo 4. Pantalla de Matricular usuarios
Postcond. si éxito	Carga diálogo modal con usuarios y roles disponibles
Postcond. si fallo	Lista vacía
Actores	Usuarios con rol profesor
Secuencia normal	1. Pulsar en el botón “Matricular”

CU-Matricula-02	Matricular usuario
Objetivo en contexto	Matricular un usuario en un grupo
Precondiciones	1. Sesión iniciada 2. Grupo seleccionado 3. Ser director o profesor en el grupo 4. Pantalla Matricular usuarios 5. Carga de lista de usuarios disponibles (CU-Matricula-01)
Postcond. si éxito	Mensaje de éxito
Postcond. si fallo	Mensajes de error
Actores	Usuarios con rol profesor
Secuencia normal	1. Seleccionar rol a asignar 2. De la lista cargada pulsar en el botón “Matricular” de la fila del usuario a matricular

CU-Matricula-03	Desmatricular usuario
Objetivo en contexto	Desmatricular un usuario de un grupo
Precondiciones	1. Sesión iniciada 2. Grupo seleccionado 3. Ser director o profesor en el grupo
Postcond. si éxito	Mensaje de éxito
Postcond. si fallo	Mensajes de error
Actores	Usuarios con rol profesor
Secuencia normal	1. Pulsar en el botón “x” de la fila del usuario a desmatricular

CU-GestObj-01	Listado profesor
Objetivo en contexto	Cargar la lista de instancias sin validar del grupo
Precondiciones	1. Sesión iniciada 2. Grupo seleccionado 3. Ser director o profesor en el grupo 4. Tipo de objeto seleccionado
Postcond. si éxito	Lista de objetos
Postcond. si fallo	Lista vacía
Actores	Usuarios con rol profesor
Secuencia normal	1. Acceder a la pantalla

CU-GestObj-02	Listado alumno
Objetivo en contexto	Cargar la lista de instancias creadas o modificadas del usuario en el grupo
Precondiciones	1. Sesión iniciada 2. Grupo seleccionado 3. Tipo de objeto seleccionado
Postcond. si éxito	Lista de objetos
Postcond. si fallo	Lista vacía
Actores	Usuarios con rol alumno
Secuencia normal	1. Acceder a la pantalla

CU-GestObj-03	Listado completo
Objetivo en contexto	Cargar la lista de instancias validadas
Precondiciones	1. Sesión iniciada 2. Grupo seleccionado 3. Tipo de objeto seleccionado
Postcond. si éxito	Lista de objetos
Postcond. si fallo	Lista vacía
Actores	Usuarios con rol profesor o alumno
Secuencia normal	1. Acceder a la pantalla

CU-GestObj-04	Eliminar instancia
Objetivo en contexto	Eliminar la instancia de un objeto
Precondiciones	1. Sesión iniciada 2. Grupo seleccionado 3. Tipo de objeto seleccionado 4. Ser profesor o alumno creador de instancia no validada 5. Aceptar el mensaje de confirmación
Postcond. si éxito	Mensaje de éxito y recarga de lista de instancias
Postcond. si fallo	Mensaje de error
Actores	Usuarios con rol profesor o alumno
Secuencia normal	1. Pulsar botón desplegable al lado de la instancia a borrar 2. Seleccionar “Eliminar” 3. Pulsar en “Borrar” en la ventana de confirmación

CU-GestObj-05	Cambiar nombre de instancia
Objetivo en contexto	Cambiar el nombre de la instancia de un objeto
Precondiciones	1. Sesión iniciada 2. Grupo seleccionado 3. Tipo de objeto seleccionado 4. Ser profesor o alumno creador de instancia
Postcond. si éxito	Mensaje de éxito y recarga de lista de instancias
Postcond. si fallo	Mensaje de error
Actores	Usuarios con rol profesor o alumno
Secuencia normal	1. Pulsar botón desplegable al lado de la instancia a renombrar 2. Seleccionar “Cambiar nombre” 3. Rellenar el campo de texto que se habilitó con el nombre 4. Pulsar el botón con el icono de guardar

CU-GestObj-06	Generar informe
Objetivo en contexto	Obtener un documento pdf con los detalles de la lista completa, con filtros avanzados aplicados o no
Precondiciones	1. Sesión iniciada 2. Grupo seleccionado 3. Tipo de objeto seleccionado
Postcond. si éxito	Visualización del documento en una nueva ventana
Postcond. si fallo	No visualización
Actores	Usuarios con rol profesor o alumno
Secuencia normal	1. Pulsar el botón “Crear informe”
Secuencia alternativa	Previo al paso 1 se puede modificar el resultado con filtros avanzados: CU-GestObj-07

CU-GestObj-07	Filtro avanzado
Objetivo en contexto	Modificar la lista completa según las instancias cumplan una lista de condiciones sobre sus atributos complejos
Dependencias	CU-GestObj-08 y CU-GestObj-09
Precondiciones	1. Sesión iniciada 2. Grupo seleccionado 3. Tipo de objeto seleccionado 4. Ser profesor o alumno
Postcond. si éxito	Recarga de lista completa
Postcond. si fallo	Mensaje de error
Actores	Usuarios con rol profesor o alumno
Secuencia normal	1. Desplegar el cuadro de filtros avanzados 2. Seleccionar en “Filtrar por” el atributo complejo. 3. Seleccionar en “Valor” la instancia de atributo complejo 4. Pulsar el botón “Filtrar con estos valores”
Secuencia alternativa	Previo al paso 4 1. Pulsar el botón con el símbolo “+” 2. Repetir los pasos 2 y 3 de la secuencia normal en la nueva fila 3. Seleccionar en la fila anterior la conjunción “AND” u “OR”

CU-GestObj-08	Listado de atributos complejos
Objetivo en contexto	Cargar de forma asíncrona la lista de atributos complejos
Precondiciones	1. Sesión iniciada 2. Grupo seleccionado 3. Tipo de objeto seleccionado 4. Ser profesor o alumno 5. Una de las dos precondiciones: 1. Haber realizado el paso 1 del caso práctico CU-GestObj-07 2. Pulsar el botón con el símbolo “+” de la secuencia alternativa
Postcond. si éxito	Desplegable “Filtrar por” con las opciones cargadas
Postcond. si fallo	Desplegable vacío
Actores	Usuarios con rol profesor o alumno
Secuencia normal	1. Automático al realizar el caso de uso CU-GestObj-07

CU-GestObj-09	Listado de instancias de atributo complejo
Objetivo en contexto	Cargar de forma asíncrona la lista de instancias de un atributo complejo
Precondiciones	1. Sesión iniciada 2. Grupo seleccionado 3. Tipo de objeto seleccionado 4. Ser profesor o alumno 5. Haber seleccionado algún valor en cualquier desplegable “Filtrar por”
Postcond. si éxito	Desplegable “Valor” con las opciones cargadas
Postcond. si fallo	Desplegable vacío
Actores	Usuarios con rol profesor o alumno
Secuencia normal	1. Automático al realizar el caso de uso CU-GestObj-07

CU-GestObj-10	Crear instancia
Objetivo en contexto	Crear una nueva instancia de un tipo de objeto
Precondiciones	1. Sesión iniciada 2. Grupo seleccionado 3. Tipo de objeto seleccionado
Postcond. si éxito	Mensaje de éxito y recarga de la lista correspondiente
Postcond. si fallo	Mensaje de error
Actores	Usuarios con rol profesor o alumno
Secuencia normal	1. Desplegar el cuadro crear 2. Introducir el nombre de la nueva instancia 3. Pulsar el botón con el símbolo “+”

CU-GestObj-11	Filtro por nombre
Objetivo en contexto	Reducir la lista de objetos mostrados a los que contengan en su nombre el texto introducido según se teclea
Precondiciones	1. Sesión iniciada 2. Grupo seleccionado 3. Tipo de objeto seleccionado
Postcond. si éxito	Ocultación de los objetos que no cumplen las condiciones
Postcond. si fallo	Mensaje de error
Actores	Usuarios con rol profesor o alumno
Secuencia normal	1. Escribir en el campo de texto “Filtrar” de cualquier lista

CU-GestObj-12	Ver instancia
Objetivo en contexto	Ver todos los atributos simples y complejos de una instancia
Precondiciones	1. Sesión iniciada 2. Grupo seleccionado 3. Tipo de objeto seleccionado
Postcond. si éxito	Ventana nueva con la información dividida en pestañas
Postcond. si fallo	Mensaje de error
Actores	Usuarios con rol profesor o alumno
Secuencia normal	1. Pulsar botón con el símbolo del ojo de la instancia a ver

CU-GestObj-13	Editar instancia
Objetivo en contexto	Modificar los atributos simples de la instancia
Precondiciones	1. Sesión iniciada 2. Grupo seleccionado 3. Tipo de objeto seleccionado 4. El objeto ha de estar validado o pertenecer al alumno que lo creó o ser un profesor del grupo
Postcond. si éxito	Editar en una ventana de edición nueva con la información dividida en pestañas y guardar el nuevo contenido
Postcond. si fallo	Mensaje de error
Actores	Usuarios con rol profesor o alumno
Secuencia normal	1. Pulsar botón desplegable al lado de la instancia a modificar 2. Seleccionar “Modificar atributos” 3. Modificar los valores que se deseen 4. Pulsar el botón “Guardar”

CU-GestObj-14	Añadir atributo complejo
Objetivo en contexto	Modificar los atributos complejos de la instancia
Dependencias	El caso de uso CU-GestObj-13 debe haberse iniciado hasta el paso 2
Precondiciones	1. Sesión iniciada 2. Grupo seleccionado 3. Tipo de objeto seleccionado 4. Edición de la instancia 5. La instancia de atributo complejo no debe estar previamente añadida
Postcond. si éxito	Añadir una nueva instancia de atributo complejo a una instancia y recarga de la lista de ese tipo de atributo complejo
Postcond. si fallo	Mensaje de error
Actores	Usuarios con rol profesor o alumno
Secuencia normal	1. Pulsar un botón con la flecha izquierda en la lista de instancias de atributo complejo disponibles
Secuencia alternativa	Si el tipo de atributo complejo tiene relación asignada, fecha o página, se llevará a cabo el CU-GestObj-15 automáticamente

CU-GestObj-15	Añadir relación
Objetivo en contexto	Añadir o editar información de relación a una instancia de atributo complejo
Dependencias	El caso de uso CU-GestObj-14
Precondiciones	1. Sesión iniciada 2. Grupo seleccionado 3. Tipo de objeto seleccionado 4. El tipo de atributo complejo tiene algún tipo de información de relación 5. Edición de la instancia
Postcond. si éxito	Añadir o modificar información de relación
Postcond. si fallo	Mensaje de error
Actores	Usuarios con rol profesor o alumno
Secuencia normal	1. Carga de ventana para editar la relación 2. Pulsar guardar
Secuencia alternativa	Si la instancia de atributo complejo existe: 1. Pulsar el botón con icono “lápiz” para editar la relación ya existente 2. Carga de ventana para editar la relación 3. Pulsar en guardar

CU-GestObj-16	Eliminar atributo complejo
Objetivo en contexto	Modificar los atributos complejos de la instancia
Dependencias	El caso de uso CU-GestObj-13 debe haberse iniciado hasta el paso 2
Precondiciones	1. Sesión iniciada 2. Grupo seleccionado 3. Tipo de objeto seleccionado 4. Edición de la instancia 5. La instancia de atributo complejo debe estar anteriormente añadida
Postcond. si éxito	Eliminar instancia de atributo complejo a una instancia y recarga de la lista de ese tipo de atributo complejo
Postcond. si fallo	Mensaje de error
Actores	Usuarios con rol profesor o alumno
Secuencia normal	1. Pulsar un botón con la flecha derecha en la lista de instancias de atributo complejo asignadas

CU-GestObj-17	Validar instancia
Objetivo en contexto	Cambiar el estado de validación
Dependencias	El caso de uso CU-GestObj-12
Precondiciones	1. Sesión iniciada 2. Grupo seleccionado 3. Tipo de objeto seleccionado 4. Ser profesor o director del grupo 5. Edición de la instancia
Postcond. si éxito	La instancia cambia de estado a validado
Postcond. si fallo	Mensaje de error
Actores	Usuarios con rol profesor
Secuencia normal	1. Pulsar el botón “Validar”

CU-GestObj-18	Validar instancia de atributo complejo
Objetivo en contexto	Cambiar el estado de validación
Dependencias	El caso de uso CU-GestObj-12
Precondiciones	1. Sesión iniciada 2. Grupo seleccionado 3. Tipo de objeto seleccionado 4. Ser profesor o director del grupo 5. Edición de la instancia
Postcond. si éxito	La instancia cambia de estado a validado
Postcond. si fallo	Mensaje de error
Actores	Usuarios con rol profesor
Secuencia normal	1. Pulsar el botón “Validar” al lado de la instancia de atributo complejo que se quiere validar

CU-GestObj-19	Escribir mensaje de validación
Objetivo en contexto	Añadir o cambiar el mensaje de validación de una instancia o de una instancia de atributo complejo
Dependencias	El caso de uso CU-GestObj-17 o CU-GestObj-18
Precondiciones	1. Sesión iniciada 2. Grupo seleccionado 3. Tipo de objeto seleccionado 4. Ser profesor o director del grupo 5. Edición de la instancia 6. Validación de la instancia o de instancia de atributo complejo
Postcond. si éxito	La instancia cambia de estado a validado o no, añadiéndose un mensaje de texto
Postcond. si fallo	Mensaje de error
Actores	Usuarios con rol profesor
Secuencia normal	1. Pulsar el botón “Validar” 2. Escribir un mensaje de texto en la ventana emergente 3. Pulsar en “Validar” o “No validar”

CU-GestObj-20	Ver mensaje de validación
Objetivo en contexto	Ver el mensaje que el profesor dejó sobre la validación o no de una instancia o instancia de atributo complejo
Precondiciones	1. Sesión iniciada 2. Grupo seleccionado 3. Tipo de objeto seleccionado 4. Ser alumno matriculado en el grupo 5. Existir un mensaje de validación, señalizado con una etiqueta azul y el número 1
Postcond. si éxito	Se muestra el mensaje de validación
Postcond. si fallo	Mensaje de error
Actores	Usuarios con rol alumno
Secuencia normal	1. Pulsar en la etiqueta “Validado” o “No validado” cuando tiene una etiqueta de menor tamaño y azul con un 1 a su derecha

CU-GestObj-21	Marcar como leído el mensaje de validación
Objetivo en contexto	Dejar constancia de que se ha leído el mensaje de validación
Dependencias	El caso de uso CU-GestObj-20
Precondiciones	1. Sesión iniciada 2. Grupo seleccionado 3. Tipo de objeto seleccionado 4. Ser alumno matriculado en el grupo 5. Existir un mensaje de validación, señalizado con una etiqueta azul y el número 1
Postcond. si éxito	La etiqueta azul con el número 1 desaparece
Postcond. si fallo	No sucede nada
Actores	Usuarios con rol alumno
Secuencia normal	1. Pulsar en la etiqueta “Validado” o “No validado” cuando tiene una etiqueta de menor tamaño y azul con un 1 a su derecha 2. Dejar seleccionado el cuadro con el texto “Marcar como leído”

CU-GestObj-22	Ver relación
Objetivo en contexto	Ver los datos que se adjuntan a una instancia de atributo complejo
Dependencias	El caso de uso CU-GestObj-12 o CU-GestObj-13
Precondiciones	1. Sesión iniciada 2. Grupo seleccionado 3. Tipo de objeto seleccionado 4. El tipo de atributo complejo tiene algún tipo de información de relación
Postcond. si éxito	Se muestra la ventana con la información
Postcond. si fallo	Mensaje de error
Actores	Usuarios con rol alumno o profesor
Secuencia normal	1. Pulsar en el botón con el icono del “ojo”

CU-Log-01	Log general
Objetivo en contexto	Ver las últimas acciones realizadas en la aplicación
Precondiciones	1. Sesión iniciada
Postcond. si éxito	Se muestra la ventana con la información
Postcond. si fallo	Mensaje de error
Actores	Usuarios con rol administrador
Secuencia normal	1. Carga automática al entrar en la ventana

CU-Log-02	Cambiar de página
Objetivo en contexto	Ver otras acciones realizadas en la aplicación que se muestran con paginación
Dependencias	CU-Log-01 o CU-GestUsu-03
Precondiciones	1. Sesión iniciada
Postcond. si éxito	Se muestra la ventana con la información y el número de página actual cambiada
Postcond. si fallo	Mensaje de error
Actores	Usuarios con rol administrador
Secuencia normal	1. Pulsar en los botones con número de página, siguiente, último, anterior o primero

CU-GestUsu-01	Crear usuario
Objetivo en contexto	Añadir un usuario a la aplicación
Precondiciones	1. Sesión iniciada 2. Pantalla de Gestión de usuarios
Postcond. si éxito	Mensaje de éxito y recarga de la lista de usuarios
Postcond. si fallo	Mensaje de error
Actores	Usuarios con rol administrador
Secuencia normal	1. Pulsar el botón con el símbolo “+” en la parte inferior 2. Rellenar el formulario correctamente 3. Pulsar en crear

CU-GestUsu-02	Editar usuario
Objetivo en contexto	Editar un usuario de la aplicación
Precondiciones	1. Sesión iniciada 2. Pantalla de Gestión de usuarios 3. El usuario existe anteriormente
Postcond. si éxito	Mensaje de éxito y recarga de la lista de usuarios
Postcond. si fallo	Mensaje de error
Actores	Usuarios con rol administrador
Secuencia normal	1. Pulsar el botón con el icono del lápiz en la fila del usuario a modificar 2. Rellenar el formulario correctamente 3. Pulsar en “Guardar cambios”

CU-GestUsu-03	Ver actividad de usuario
Objetivo en contexto	Ver la actividad de un usuario concreto en la aplicación
Precondiciones	1. Sesión iniciada 2. Pantalla de Gestión de usuarios 3. El usuario existe anteriormente
Postcond. si éxito	Carga la página del log de usuario
Postcond. si fallo	Mensaje de error
Actores	Usuarios con rol administrador
Secuencia normal	1. Pulsar el botón con el icono de la lupa en la fila del usuario

CU-GestUsu-04	Borrar usuario
Objetivo en contexto	Eliminar un usuario. Acción muy restringida dado que se prefiere cambiar el rol a inactivo
Precondiciones	1. Sesión iniciada 2. Pantalla de Gestión de usuarios 3. El usuario existe anteriormente 4. El usuario no es creador de ninguna instancia de objeto 5. El usuario no está matriculado en ningún grupo como profesor o alumno 6. El usuario no tiene actividad en el log
Postcond. si éxito	Mensaje de éxito
Postcond. si fallo	Mensaje de error
Actores	Usuarios con rol administrador
Secuencia normal	1. Pulsar el botón con el icono del aspa en la fila del usuario

7.3 Anexo III. Acrónimos

- AJAX: Asynchronous JavaScript And XML
- DAO: Data Access Object
- JSON: JavaScript Object Notation
- JSP: Java Server Pages
- JSTL: JSP Standard Tag Library
- LMS: Learning Managment System
- PHP: PHP Hypertext Preprocessor