UNIVERSITAT OBERTA DE CATALUNYA

Estudios de Informática, Multimedia y Telecomunicaciones

PROYECTO FIN DE CARRERA

Ingeniería en Informática

Programa de gestión de juegos para la plataforma k-PAX

Alumno: Rubén Viguera Marañón

Directores: Jordi Duch Gavaldà

Heliondo Tejedor Navarro

Logroño, a 26 junio de 2012

Índice

Índice de contenido

Introducción

El proyecto surge como un proyecto de mejora sobre la plataforma k-PAX. El objetivo es incorporar a la plataforma un módulo de gestión de videojuegos que ofrezca servicios de búsquedas de juegos por diferentes categorías o características o el filtrado sobre dichas búsquedas según diferentes criterios.

k-PAX es una plataforma de aprendizaje en red que surge como un proyecto de innovación docente, es decir, una plataforma en línea que permite el aprendizaje en red mediante juegos. k-PAX se basa en cuatro puntos:

1. el apoyo al trabajo y la evaluación de ciertas competencias mediante el aprendizaje basado en JUEGOS.
2. El estado actual de la tecnología, que añade una nueva dimensión al sincronía espacial adicional de la UOC gracias a los nuevos dispositivos portátiles que incorporan la calle, el transporte público, etc. a los lugares donde la comunidad UOC puede estar conectada y aprendiendo.
3. Esta misma tecnología ofrece múltiples herramientas pues 2.0, como son las REDES SOCIALES. Éstas permiten añadir relaciones virtuales a cualquier hecho que pueda ocurrir en un ordenador personal, tablet, etc.
4. los resultados de trabajo de innovación que se hace en la UOC muchas veces quedan limitados a una asignatura, a unos estudios, etc. Cuando es muy posible que estos sean útiles y aplicables a otros ámbitos.

k-PAX es una plataforma ya funcional que aporta actualmente los servicios básicos. La creación de este proyecto supone para k-PAX poder disponer de un módulo de búsqueda de videojuegos inexistente hasta el momento, aunque debido a la poca cantidad de videojuegos incorporados no se trate de una funcionalidad inmediatamente necesaria. Sin embargo, sí que lo será a medio plazo.

Documento de Objetivos del Proyecto

Objetivos

El objetivo fundamental del PFC es la elaboración de un nuevo módulo compatible con la plataforma k-PAX que permita a los usuarios de la plataforma realizar toda una serie de búsquedas sobre los videojuegos disponibles en la misma, así como poder filtrar dichas búsquedas según diferentes criterios.

Descripción

Se trata de realizar una aplicación de escritorio, que sea capaz de:

1. proporcionar un servicio de búsquedas de juegos:
    - por categoría.
    - Por nombre.
    - Similares a otro.
2. Ordenar una búsqueda por:
    - los más jugados.
    - Los más populares o votados.
    - Los más nuevos.
    - Los más comentados.
    - Los más activos

Adicional y opcionalmente, si el tiempo lo permite, se añadirán algunas o todas las funciones siguientes:

- Clasificar juegos por medio de un sistema de gestión de etiquetas (tags)
- Subir enlaces de juegos y editar sus metadatos.

Antecedentes

k-PAX es una plataforma existente, que actualmente ofrece los servicios mínimos. Permitiendo la gestión básica de los videojuegos, usuarios y administración de la plataforma. carece, sin embargo, con ninguna funcionalidad de búsqueda de videojuegos. Servicio que se añadirá con la realización de este proyecto.

k-PAX es una plataforma cliente-servidor de software libre cuyo código está alojado en la web de github, donde se puede encontrar tanto la parte cliente como la servidora. Este proyecto se realizará a partir de una bifurcación (fork) de la rama principal (branch) disponible en la web.

Alcance

Esta aplicación será utilizada tanto por usuarios comunes, que podrán ser alumnos o no de la UOC, como por administrativos que podrán realizar actividades de redes sociales, jugar aprendiendo y/o administrar su propio perfil o la plataforma en sí misma.

La aplicación debe cumplir los siguientes requisitos funcionales:

1. Ha de permitir realizar búsquedas sobre el conjunto de videojuegos disponibles en la plataforma k-PAX según diferentes criterios tales como búsqueda por categorías, por nombre o por similitud.
2. Su diseño deberá permitir la incorporación de nuevos criterios de búsqueda.
3. De forma complementaria, una búsqueda deberá poder ser ordenada según diferentes criterios tales como los más jugados, los más populares o votados, los más nuevos, los más comentados o los más activos.
4. Igualmente, su diseño deberá permitir la incorporación de nuevos criterios de ordenación.
5. Implementar los cambios de el proyecto de modo que pueda ser probado posteriormente tanto por funciones automáticas (como por ejemplo JUnit) como por un grupo de personas (en este caso un grupo de alumnos de matemáticas)
6. Desarrollar la memoria de modo que pueda ser consultada por proyectistas posteriores sirviendo de documentación técnica que informe de forma detallada los cambios realizados, de modo que puedan apoyarse en dicha documentación para realizar sus proyectos.
7. Seguir un diseño extensible, abierto y sencillo que mantenga la estabilidad y robustez de la plataforma.
8. Se podrán clasificar juegos por medio de un sistema de gestión de etiquetas (tags).
9. Se podrán realizar subidas de enlaces de juegos y editar sus metadatos.

Se deberá entregar el código resultante del proyecto, que podrá ser realizado por medio de una o varias subidas a los proyectos de la plataforma disponibles en github, así como una memoria que detalla el desarrollo del proyecto. No será necesario ningún tipo de instalador, ni de manual de usuario, aunque no se descartan.

En caso de necesitar eliminar funcionalidades al programa para su finalización en la fecha prevista o por ser demasiado dificultoso, se admite no desarrollar los puntos 8 y 9, los cuales son opcionales.

Tecnología y arquitectura

k-PAX se caracteriza por utilizar numerosas tecnologías y por tener muchas propiedades que deberán ser mantenidas durante el desarrollo y posible posterior incorporación del módulo resultante de este proyecto, es decir, que la incorporación a la rama principal de k-PAX del módulo que resulte de este proyecto, no deberá afectar a las propiedades, características y funcionalidades propias de la plataforma, sino únicamente añadir nuevas.

Entre las propiedades y características propias de k-PAX, destacan las siguientes:

- Es una plataforma separada en tres componentes:
  - La parte de los clientes, que pueden ser juegos, aplicaciones para mostrar resultados, estadísticas, modificar perfiles de usuarios, conectarse a la red, etc. Aquí también se incluyen todos esos PFCs que sean juegos. Esta parte para comunicarse con el resto de la plataforma usa llamadas de servicios que ya están implementados (o que se van a implementar en PFCs como los vuestros).
  - El core de la aplicación (modelo+controlador), que es lo que has comentado tu. Permite

que la plataforma elgg sea transparente (que es donde los usuarios se conectan) y también gestiona la base de datos. En esta parte hay algunos servicios implementados, como por ejemplo registrar un juego, el control de las puntuaciones, etc. Como bien has dicho, todo este servicio esta montado sobre Java y es el proyecto que esta en el github.
    - Finalmente, la parte de visualización, que se implementa mediante módulos o plugins de elgg. Estos plugins se activan dentro de la plataforma, y su función principal es leer (o enviar) datos a la base de datos (los módulos del punto anterior) para presentarla al usuario (y recoger sus acciones).
- El core de la aplicación esta desarrollada en lenguaje de programación Java.
- La capa de persistencia de la plataforma es accedida por medio del ORM Hibernate.
- Utiliza la librería Apache Maven para la gestión y configuración de la aplicación.
- Utiliza un servidor de aplicaciones JBoss.
- Utiliza un sistema gestor de base de datos MySQL en su versión 5.
- Está diseñada de modo que pueda ser utilizada por cualquier sistema operativo y cualquier dispositivo. Es importante mantener esta característica y no ligar el proyecto a un sistema concreto.

Recursos humanos y comunicación

El desarrollo de la aplicación y la redacción de esta memoria serán competencia del proyectante.

Existirá la figura del director de proyecto, que se encargará de:

1. Dirigir al proyectante y evitar que se desvíe de los objetivos.
2. Se asegurará de que se cumplen los plazos.
3. Revisará la estructura de la memoria, asegurando que ésta se ajusta a un modelo estándar.

El beneficiario directo del módulo será el usuario final de la página web de k-PAX, así como la comunidad opensource que podrá disponer de su implementación desde la web de github, una vez se haya realizado la subida definitiva, ya sea para su incorporación en la rama principal de k-PAX o en cualquiera de sus bifurcaciones (forks).

El proyectante es D. Rubén Viguera Marañón, alumno de la Universitat Oberta de Catalunya, mientras que la figura de tutor de proyecto recae sobre D. Jordi Duch Gavaldà y sobre D. Heliondo Tejedor Navarro, consultores de la Facultad de Estudios de Informática, Multimedia y Telecomunicaciones de la misma universidad, respectivamente. Por el contrario, no existe una figura representativa del cliente, que recae en el usuario final de k-PAX.

La comunicación, puesta en común y seguimiento de resultados se realizará mediante correo electrónico. No será posible la realización de reuniones presenciales debido a la naturaleza de estudios a distancia que caracteriza a esta universidad.

Metodología

La diferencia de importancia y carácter opcional de algunos de los requisitos del sistema, han inducido a pensar en desarrollar el proyecto con una metodología basada en un plan de fases incrementales, en donde cada una de ellas pretenda añadir y en su caso solucionar funcionalidades encontradas en la fase anterior y que darán lugar a diferentes versiones de la aplicación.

De entre las posibles metodologías que admiten ciclos de vida incrementales, se ha decidido basarse en la del proceso unificado del desarrollo de software por ser una metodología bien diseñada para proyectos con el ciclo de vida anteriormente expuesto.

Cada una de dichas fases se compondrá de las fases habituales de análisis, diseño, implementación y pruebas.

Esta metodología es acertada en un proyecto de índole modular como éste, ya que cuenta con algunos requisitos que han de estar presentes en una primera versión del programa siendo pospuestos otros requisitos menos importantes para versiones posteriores pudiéndose incluso carecer de ellos.

Un sistema de fases incrementales permite establecer objetivos intermedios, de manera que en el caso de que llegase la fecha límite a su fin, el proyecto podrá quedar más o menos cerrado y definido en un punto concreto, facilitando así su continuación por otro proyectante.

Riesgos y planes de acción

Durante la realización del proyecto, existirán una serie de riesgos que pueden retrasar su desarrollo o incluso obligar al proyectante a entregar una versión del programa que no cumpla todos los requisitos opcionales.

Estos riesgos son:

1. Actividades extras ajenas al proyecto: El proyecto se llevará a cabo en paralelo con la vida laboral del proyectante que realiza en horario laboral hasta la media tarde, y con otras tres asignaturas de la UOC, provocando que haya épocas en las que el proyecto avance más despacio.
    a. *Probabilidad*: Seguro.
    b. *Momento previsto*: Desde el inicio del proyecto hasta su finalización en Junio de 2012.
    c. *Plan de contingencia*: Tomar este horario en consideración durante la estimación del proyecto.
2. El hecho de tener que cuadrar las agendas de proyectante y director pueden dar lugar a retrasos de varios días:
    a. *Probabilidad*: Alta.
    b. *Momento previsto*: Durante el desarrollo del proyecto.
    c. *Plan de contingencia*: Planificar el proyecto con tareas que se puedan realizar de forma paralela, para adelantar en las tareas no dudosas.
3. Errores de diseño, omisiones o malentendidos: La poca experiencia en cuanto al diseño y análisis de programas por parte del proyectante pueden dar lugar a cambios en los requisitos del sistema.
    a. *Probabilidad*: Alta.
    b. *Momento previsto*: Durante el desarrollo del proyecto.
    c. *Plan de contingencia*: Se remite al plan de contingencia de cambios en los requisitos del sistema descrito anteriormente.

4. Estimaciones mal realizadas o poco realistas: Normalmente producidas por la poca experiencia del proyectante para estimar tiempos no coincidiendo el tiempo estimado con el real para realizar las tareas o que puedan provocar ralentizaciones en el desarrollo del proyecto.
    a. *Probabilidad*: Segura.
    b. *Momento previsto*: Durante el desarrollo del proyecto.
    c. *Plan de contingencia*: Realizar la estimación con técnicas de estimación, tales como el cálculo de puntos función y su trasformación a horas de trabajo necesarias. Además, si el tiempo real es mayor que el estimado se ha de aplicar el plan de contingencia por defecto explicado más abajo, en caso contrario se continuará con el plan de tareas del proyecto y si se ha decidido, anteriormente, eliminar algún incremento opcional, se deberá volver a estudiar la posibilidad de añadirlo de nuevo al plan de desarrollo.
5. Falta de recursos: producidas por la no disposición de recursos para la consecución efectiva de ciertas tareas del proyecto, por ejemplo que se estropee el ordenador principal desde el que se desarrolla el proyecto.
    a. *Probabilidad*: Muy baja.
    b. *Momento previsto*: Durante el desarrollo del proyecto.
    c. *Plan de contingencia*: Una vez identificados los recursos necesarios buscar duplicados de los mismos, en caso de no encontrar uno, pactar con el director las medidas necesarias.
6. Pérdida de información o de archivos: Una caída de luz, un sector dañado del disco en el que se almacena el proyecto o confusiones con versiones del programa pueden dar lugar a cierta pérdida de información.
    a. *Probabilidad*: Medio.
    b. *Momento previsto*: Durante el desarrollo del proyecto.
    c. *Plan de contingencia*: Realizar subidas frecuentes al fork en Github del proyecto.
7. Ausencia del proyectante por causas justificadas: Enfermedades, trabajo, etc.
    a. *Probabilidad*: Media.
    b. *Momento previsto*: Durante el desarrollo del proyecto y en especial en temporada de bajas temperaturas o en situaciones de crisis y de estrés continuado en el proyectante.
    c. *Plan de contingencia*: El de por defecto explicado más abajo.

Existe un plan de contingencia por defecto que consiste en hacer las estimaciones de tiempo tomando únicamente los días laborables de lunes a viernes, y en caso de producirse un retraso utilizar las mañanas y tardes de los Sábados y mañanas de los domingos durante como máximo dos semanas. Si durante ese tiempo no se ha podido salir de la crisis, se buscará una solución diferente para el problema que cause el retraso.

En caso de que exista un retraso de un mes en cuanto a la estimación inicial, se retirará del proyecto el incremento opcional menos importante hasta el momento, pudiéndose reincorporar dicho incremento al proyecto en caso de que se reduzca considerablemente el retraso de tiempo.

Análisis

Descripción general del proceso

El proyectante deberá implementar un módulo nuevo compatible con k-PAX que permita realizar búsquedas de los juegos existentes en la plataforma, así como ordenaciones sobre dichas búsquedas.

Al término del proyecto, el usuario final de k-PAX será capaz de realizar las siguientes acciones:

1. Realizar búsquedas sobre el conjunto de videojuegos disponibles según diferentes criterios. Estos son:
    a. Búsqueda por categorías
    b. Búsqueda por nombre.
    c. Búsqueda por similitud.
2. Implementar e incorporar nuevos criterios de búsqueda.
3. Ordenar una búsqueda según criterios tales como:
    a. Los más jugados.
    b. Los más populares o votados.
    c. Los más nuevos.
    d. Los más comentados.
    e. Los más activos, es decir, aquellos para los que están jugando más personas actualmente.
4. Implementar e incorporar nuevos criterios de ordenación.
5. Opcionalmente, y si el tiempo lo permite, clasificar juegos por medio de un sistema de gestión de etiquetas (tags).
6. Opcionalmente, y si el tiempo lo permite, realizar subidas de enlaces de juegos y editar sus metadatos.
7. k-PAX mantendrá su independencia del sistema operativo y del dispositivo en la que se ejecute o desde la que se acceda, más allá de lo dependiente que ya sea.

Para cumplir los requisitos descritos, el proyectante tendrá que desarrollar una serie de entregables entre la que se destaca el código de la aplicación resultante del proyecto, un manual de instalación y un script SQL con los cambios realizados en la estructura de base de datos actual.

Actores del sistema

Se distinguen en el sistema distintos tipos de actores, es decir, que las personas que utilizarán la aplicación interpretan distintos roles en el sistema.

Para controlar el acceso a las distintas partes o funciones de la aplicación, se establecerán diferentes privilegios o derechos de uso de las funcionalidades del sistema.

Los tipos de rol o actores que se presentan en el sistema son los siguientes:

Administrador

Identificación
Representa a todo usuario que tiene control total del sistema y acceso a todos los datos críticos o no, que maneja el programa, es decir, es el que mayores privilegios tiene en el sistema.

Contexto
Generalmente será la persona o personas creadoras de la plataforma las que se encarguen de elegir la persona que realice las labores de administración.

Objetivos
El administrador será quien tenga control total sobre el sistema, es decir, tiene todos los privilegios que se le puedan conceder y no habrá ningún otro rol que tenga más o diferentes permisos a él.

Su labor será la de gestionar los datos de la plataforma, tales como gestión de juegos (alta, baja, modificación de sus datos, permitir su acceso a grupos), gestión de grupos, gestión de usuarios, realización de estadísticas o informes, etc. También será el que se encargue de conceder permisos a grupos, para poder acceder a un juego o de relevar el mando en otro administrador y por esta razón siempre deberá haber al menos un administrador.

En lo que atañe a este proyecto, podrán realizar modificaciones en los metadatos de los juegos, realizar subidas de enlaces a juegos y asignar etiquetas a juegos.

Usuario común

Identificación
Representa a todo usuario del sistema. Puede jugar juegos, realizar actividades de redes sociales (comentar, buscar amigos) y en este caso buscar juegos.

Contexto
Es el perfil básico de todo usuario del sistema. Todos los usuarios de k-PAX son usuarios comunes.

Objetivos
Los usuarios comunes serán capaces de realizar búsquedas sobre los juegos a los que tengan acceso, así como ordenar las búsquedas, votar un juego positiva o negativamente, jugar juegos, realizar comentarios, etc.

En lo que atañe al desarrollo de este módulo, sólo los usuarios pertenecientes a g

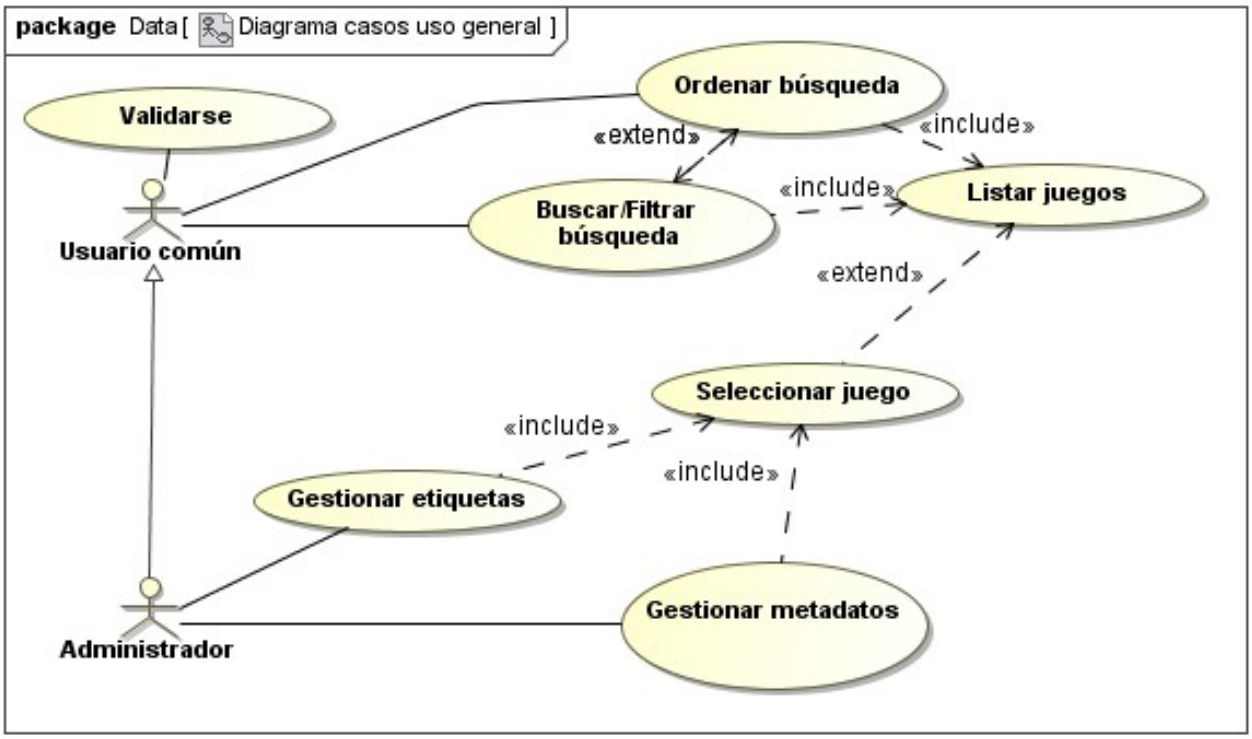
Modelo de casos de uso

En este apartado se muestra el modelo de casos de uso del proyecto, que permite analizar las funcionalidades que podrá realizar cada actor. Para facilitar el dibujo y comprensión de los casos de uso en que se descompone el proyecto, se ha optado por agruparlos en las llamadas explosiones de casos de uso.

Una explosión de casos de uso no es sino un caso de uso que agrupa según una funcionalidad más amplia a varios casos de uso. Cada explosión de caso de uso se dibujará en el diagrama de casos de

uso general y los casos de uso propios de cada explosión se incluirán en un diagrama de caso de uso separado del mismo nombre que su explosión de caso de uso.

En el diagrama principal se indican todas las explosiones de casos de uso y otros casos de uso independientes:

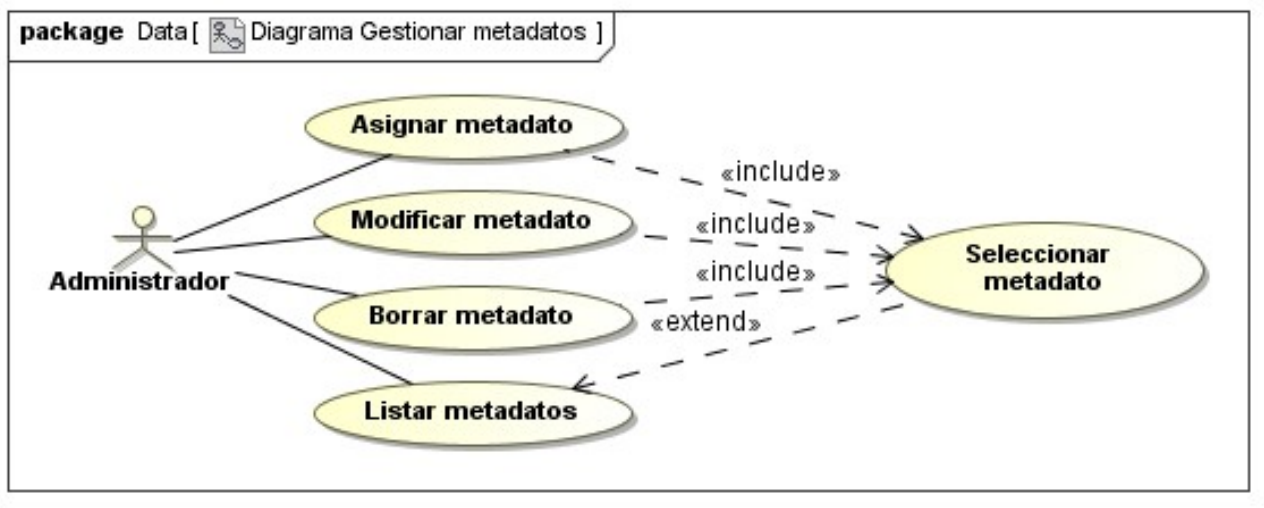


Modelo de casos de uso general

1. *Validarse:* Consiste en identificar y autenticar a un usuario en el sistema y permite el uso de distintas funciones de la aplicación.
2. *Buscar/Filtrar búsqueda:* Establece un criterio de búsqueda/filtrado a elegir. Posteriormente muestra el listado de juegos que cumplen los criterios especificados. Puede complementarse con un criterio de ordenado (ver caso de uso siguiente).
3. *Ordenar búsqueda:* Establece un criterio de ordenación a elegir. Posteriormente muestra el listado de juegos que cumplen los criterios especificados. Puede complementarse con un criterio de búsqueda/filtrado (ver caso de uso anterior).
4. *Listar juegos*: Muestra un listado de juegos que cumplen ciertos criterios -ver los dos casos de uso anteriores-. Es posible seleccionar cada uno de ellos para consultar o modificar su ficha, jugarlo, etc.
5. *Seleccionar juego:* Muestra la ficha de un juego y, en el caso de que el actor sea un administrador, también permitirá modificar sus datos. Para seleccionar un juego, primero es necesario buscarlo y listarlo.
6. *Gestionar etiquetas*: Permite eliminar, modificar o asociar palabras a un juego, que en su conjunto describen las características del juego por medio de conceptos o etiquetas. Esta información será la base de ciertos criterios de búsqueda u ordenación de juegos. La gestión de etiquetas está ya implementada.
7. *Gestionar metadatos:* Permite eliminar, modificar o asociar metadatos a un juego (un término con su valor). Esto permite indicar etiquetas que ayuden a describir el juego. Por ejemplo, el título, genero, edades recomendadas, lenguaje, tags, valoraciones, etc...

A continuación, se describen las explosiones de casos de uso junto con los casos de uso que agrupa y que se obvian en el diagrama general de casos de uso.

Gestionar metadatos



Administra todo lo referente a los metadatos de un juego. Un metadatos es una palabra clave o término no jerárquico asignado a un juego, junto con un valor. Un metadato ayuda a describir un juego. Los metadatos son establecidos por los administradores, generalmente escogidos de forma informal o personal.

Cualquiera de los casos de uso de esta explosión de casos de uso requiere haber seleccionado primero un juego sobre el que gestionar sus metadatos.

1. Asignar metadato: Añade o asigna un nuevo metadato a un juego.
2. Modificar metadato: Modifica la descripción del metadato asociado a un juego.
3. Listar metadatos: Muestra una lista con los metadatos asociados a un juego.
4. Borrar metadato: Elimina o desvincula un metadato a un juego.
5. Seleccionar metadato: Selecciona un metadato para poder modificarla o borrarla.

Para no complicar los diagramas de casos de uso se ha obviado la relación de inclusión del caso de uso “Validar usuario” con los demás, que es necesario previamente a cualquier otro caso de uso.

Requisitos del sistema

Para poder llegar a generar una aplicación que cubra el problema planteado, se deben conocer primero todos los requisitos específicos que deberá cumplir la aplicación.

Esos requisitos se enumeran y describen a continuación, para una mejor referencia a ellos:

R1

El sistema deberá disponer de un sistema de identificación y autenticidad de usuarios, que se supone ya existente. Si el usuario se identifica y autentica correctamente, el sistema deberá permitir el acceso a la aplicación del sistema a ese usuario. Toda funcionalidad desarrollada en este proyecto requiere que el usuario haya realizado una sesión y que ésta continúe abierta, en el momento de su ejecución.

R2
El sistema deberá implementar un sistema de permisos, que se supone ya existente, que determine si un usuario interpreta el rol de administrador o de usuario común para poder permitir o denegar la ejecución de las funcionalidades desarrolladas en este proyecto.

R3
Todo usuario podrá listar; y en consecuencia filtrar, ordenar y seleccionar, juegos, única y exclusivamente, de entre de entre el conjunto de juegos a los que tenga acceso.

R4
Un usuario con los derechos suficientes, debe poder acceder a la ficha de un juego para la visualización de sus datos (esto es, modificar sus metadatos y sus etiquetas). En usuarios comunes, este permiso de acceso no implica poder modificar o borrar todos o parte de los datos, lo cual está reservado a los administradores, sino únicamente poder visualizar los datos.

Especificación detallada de cada Caso de Uso

A continuación se detallan los casos de uso anteriores.

Caso de Uso 1. Validar Usuario

Actores
Todos.

Descripción
Representa el inicio de una nueva sesión en la aplicación. Es un mecanismo de seguridad de identificación y autenticidad que impide el acceso a la aplicación a usuarios no autorizados.

Consiste en la introducción de un identificador y una clave de acceso, que en caso de ser validos para el sistema, dará acceso al usuario a un panel de control con los permisos correspondientes al rol que tenga relacionado con su identificador.

Todo identificador ha de tener asociado un perfil de privilegios. La identificación correcta del usuario permite a la aplicación conocer sus privilegios y autorizarle el uso de todas o parte de las funciones de la aplicación.

Este caso de uso, se considera implementado y no se trabaja en este proyecto.

Poscondición
El usuario se valida, y se le muestra un panel con acceso a todas o algunas de las partes de la aplicación.

Secuencia normal

1. El usuario accede a la página de login de la plataforma k-PAX.
2. El sistema solicita el identificador y la clave de acceso.
3. El usuario las facilita.
4. El sistema comprueba la identidad del usuario y su autenticidad.
5. El sistema muestra un panel de control diferente en relación a los permisos asociados al

identificador.

Frecuencia
Una vez por sesión.

Importancia
Muy baja, puesto que no se trabaja en este proyecto.

Urgencia
Ninguna.

Requisitos
R1, R2.

Caso de Uso 2. Buscar/Filtrar búsqueda

Actores
Todos.

Descripción
El sistema muestra un conjunto de criterios de búsqueda, de entre los cuales el usuario deberá escoger uno y en su caso solicita datos necesarios para ejecutar la búsqueda de videojuegos.

Precondición
Caso de uso 1

Poscondición
Los criterios seleccionados quedan fijados para un listado de juegos posterior.

Secuencia normal

1. El sistema muestra los criterios de búsqueda posibles.
2. El actor indica cuál de los criterios de búsqueda desea utilizar.
3. El sistema, dependiendo del criterio de búsqueda escogido, solicita los datos necesarios para realizar la búsqueda con dicho criterio.
4. El actor aporta los datos necesarios. Si no es necesario ningún dato complementario para ejecutar la búsqueda se pasa automáticamente al punto siguiente.
5. El sistema guarda la preferencia

Frecuencia
Normal-Alta.

Importancia
Normal.

Urgencia
Normal.

Requisitos
R3

Caso de Uso 3. Ordenar búsqueda

Actores
Todos.

Descripción
El sistema muestra un conjunto de criterios de ordenado, de entre los cuales el usuario deberá escoger uno y en su caso solicita datos necesarios para ejecutar la ordenación de videojuegos.

Precondición
Caso de uso 1

Poscondición
Los criterios seleccionados quedan fijados para un listado de juegos posterior.

Secuencia normal

1. El sistema muestra los criterios de ordenación posibles.
2. El actor indica cuál de los criterios de ordenación desea utilizar.
3. El sistema, dependiendo del criterio de ordenación escogido, solicita los datos necesarios para realizar la ordenación con dicho criterio.
4. El actor aporta los datos necesarios. Si no es necesario ningún dato complementario para ejecutar la búsqueda se pasa automáticamente al punto siguiente.
5. El sistema guarda la preferencia.

Frecuencia
Normal-Alta.

Importancia
Normal.

Urgencia
Normal.

Requisitos
R3

Caso de Uso 4. Listar juegos

Actores
Todos.

Descripción
El sistema muestra un listado de juegos que cumplen los criterios de búsqueda/filtrado y, de entre los cuales el usuario deberá escoger uno. Es posible realizar posteriormente el realizar el caso de uso *seleccionar juego*, seleccionando un juego de dicha lista.

Precondición

Caso de uso 1, 2 y 3

Secuencia normal

1. El sistema busca los juegos a los que tiene acceso el usuario, restringiendo el resultado según los criterios de búsqueda/filtrado y de ordenación indicados anteriormente.
2. El sistema muestra de forma visual el listado de juegos encontrados.
3. Si lo desea, el actor puede seleccionar de entre los juegos listados, uno de ellos, realizando el caso de uso seleccionar juego.

Frecuencia
Normal-Alta.

Importancia
Normal.

Urgencia
Normal.

Requisitos
R3

Caso de Uso 5. Seleccionar Juego

Actores
Todos

Descripción
El actor selecciona de entre la lista de juegos aquel con el que quiere trabajar, el juego seleccionado hasta el momento se cambia por el nuevo.

Precondición
Casos de uso 1 y 4.

Poscondición
El juego especificado queda seleccionado como el juego con el que se trabaje al finalizar el caso de uso.

Secuencia normal

1. El actor selecciona de la listad e juegos listados, aquel con el que quiere trabajar.
2. El sistema cambia el juego que se utilizaba hasta el momento por el seleccionado.

Excepciones
No se puede seleccionar más de un juego.

Frecuencia
Normal.

Importancia
Normal.

Urgencia
Normal.

Requisitos
R4.

Caso de Uso 6. Asignar metadato

Actores
Administrador.

Descripción
El actor proporciona el metadato (palabra o término) con su valor. Finalmente se guarda la información como un nuevo metadato asociado al último juego seleccionado.

Precondición
Caso de uso 1, 5

Secuencia normal

1. El actor solicita añadir un metadato a un juego.
2. El sistema muestra un formulario para introducir el metadato.
3. El actor indica el nuevo metadato.
4. Si todo es correcto, se almacena el metadato, asociándolo al juego.

Frecuencia
Baja.

Importancia
Normal.

Urgencia
Normal.

Requisitos
R4

Caso de Uso 7. Listar metadatos

Actores
Administrador.

Descripción
El sistema muestra todas los metadatos asociados al último juego seleccionado en una lista ordenada. Es posible realizar el caso de uso *seleccionar metadato*, seleccionando una etiqueta de dicha lista.

Precondición
Caso de uso 1 y 5.

Secuencia normal

1. El actor solicita la lista de metadatos del juego seleccionado.
2. El sistema busca todos los metadatos asociados al juego.
3. El sistema ordena los metadatos por nombre y los muestra en una lista.
4. Si el actor lo desea puede realizar el caso de uso *seleccionar metadato*, seleccionando un metadato de dicha lista.

Frecuencia
Cada vez que se visualice la ficha de un juego.

Importancia
Media.

Urgencia
Media-Alta.

Requisitos
R4.

Caso de Uso 8. Seleccionar Metadato

Actores
Todos

Descripción
El actor selecciona, de entre la lista de metadatos del juego seleccionado, aquel con el que quiere trabajar. El metadato seleccionado hasta el momento se cambia por el nuevo.

Precondición
Casos de uso 1 y 5.

Poscondición
El metadato especificado queda seleccionado como el metadato con la que se trabaje al finalizar el caso de uso.

Secuencia normal

3. El actor selecciona de la lista de metadatos listados asociados al juego seleccionado, aquel con el que quiera trabajar.
4. El sistema cambia el metadato que se utilizaba hasta el momento por el seleccionado.

Excepciones
No se puede seleccionar más de un metadato.

Frecuencia

Normal.

Importancia
Normal.

Urgencia
Normal.

Requisitos
R4.

Caso de Uso 9. Borrar metadato

Actores
Administrador.

Descripción
Elimina un metadato asociado a un juego.

Precondición
Caso de uso 1, 5 y 8.

Secuencia normal

1. El actor da la orden de eliminar el metadato seleccionado.
2. El sistema comprueba si el usuario tiene permiso para eliminar el metadato al juego.
3. El sistema busca el metadato indicado para el juego seleccionado, y lo borra definitivamente.

Frecuencia
De forma ocasional y sobre todo al añadir un juego.

Importancia
Media.

Urgencia
Media.

Requisitos
R4.

Caso de Uso 10. Modificar metadato

Actores
Administrador.

Descripción
El actor indica el nuevo metadato del juego seleccionado, modificando el término anterior por el nuevo.

Precondición
Caso de uso 1, 5 y 8.

Secuencia normal

1. El actor solicita cambiar el metadato seleccionado.
2. El sistema comprueba si el actor tiene derecho para cambiar el metadato del juego.
3. El actor indica el nuevo metadato.
4. Si todo es correcto, el sistema modifica el metadato anterior por el nuevo.

Frecuencia
Muy baja.

Importancia
Normal.

Urgencia
Aplazable si se tiene el caso de uso 6 y 9, sino normal.

Requisitos
R4.

Análisis de clases

La aplicación debe conectarse a una BD, concretamente a un SGBD MySQL en su versión 5, por medio de un ORM denominado Hibernate y para el que ya existe una capa de persistencia que facilita su independencia de la capa de negocio y la de presentación.

También se ha indicado en el apartado de Tecnología y Arquitectura que la plataforma se ejecuta en un entorno Web cliente-servidor. Es por ello que por diseño se ha implementado mediante un sistema en capas.

Se definen entonces 3 capas:

1. Persistencia: Encargada de hacer la conexión con la base de datos y consultar y guardar los datos de los objetos de una clase en una o varias tablas de la base de datos.
2. Lógica: Encargada de comprobar los datos que introduce el usuario y hacer de intermediario entre la capa de persistencia y la de presentación.
3. Presentación: Encargada de mostrar los datos de la lógica de negocio al usuario de la forma más simple e intuitiva y de entorno de entrada de datos hacia el sistema.

La capa de persistencia actual posee un conjunto de tablas que corresponden a diferentes entidades. Estas son: usuario (user) -que representa a un usuario-, Realm – que representa los alias de los usuarios para diferentes páginas desde las que se puede acceder a la plataforma, tales como Facebook o UOC, grupo (group) -que indica los juegos a los que tiene acceso un usuario-, los juegos (game), las instancias de juegos -que indican los juegos a los que está jugando un usuario-, GameLike, que indica si a un usuario le gusta un juego, así como el resto de tablas que relacionan las entidades entre sí.

Actualmente cada entidad se corresponde con una clase y esta a su vez con una tabla en la base de datos. En la página siguiente se puede ver el diagrama de tablas de la base de datos, actual. Este diagrama sólo muestra el nombre de las tablas y sus principales datos, esto es que no se muestran todos los datos de cada tabla. Sobre las tablas existentes será necesario añadir aquellas tablas o campos que sean necesarios para llevar a cabo las funcionalidades que se desarrollarán en este proyecto. Las clases que se definan a continuación, pertenecerán a la capa de lógica. El resto de clases del sistema se identificarán en la fase de diseño.

A continuación se muestra el listado de las entidades que se utilizarán en el proyecto. Muchos de ellos tendrán que ser añadidos o incluso modificados por medio de clases Java y tablas en BD para poder dar soporte al nuevo módulo a desarrollar.

1. **Tag (etiqueta)**: Clase que guarda la información de una etiqueta y el juego al que está asignado. Elgg proporciona ya la gestión de etiquetas.

   Interviene principalmente en los casos de uso de la explosión de casos de uso "Gestionar etiquetas" y para el de Buscar/Filtrar juego por similitud.

2. **Game (Juego):** Es necesaria la existencia de algún campo con el nombre del juego para poder realizar búsquedas/filtrados de juegos por medio de su nombre. Este dato ya existe con el nombre de *name* en la tabla *Game.*

   También será necesaria la existencia de algún campo que indique la *categoría* del juego (acción, habilidad, lógica, multijugador, rol, carreras, plataformas, etc.). En este caso no existen ningún campo que indique tal, por lo que será necesario añadir uno. Este campo será utilizado para realizar búsquedas de juegos *por categoría*.

   Igualmente será necesaria la existencia de algún campo que indique la *fecha de subida* del juego. En este caso no existen ningún campo que indique tal, por lo que será necesario añadir uno. Este campo será utilizado para realizar ordenaciones de búsquedas de juegos por *los más nuevos*.

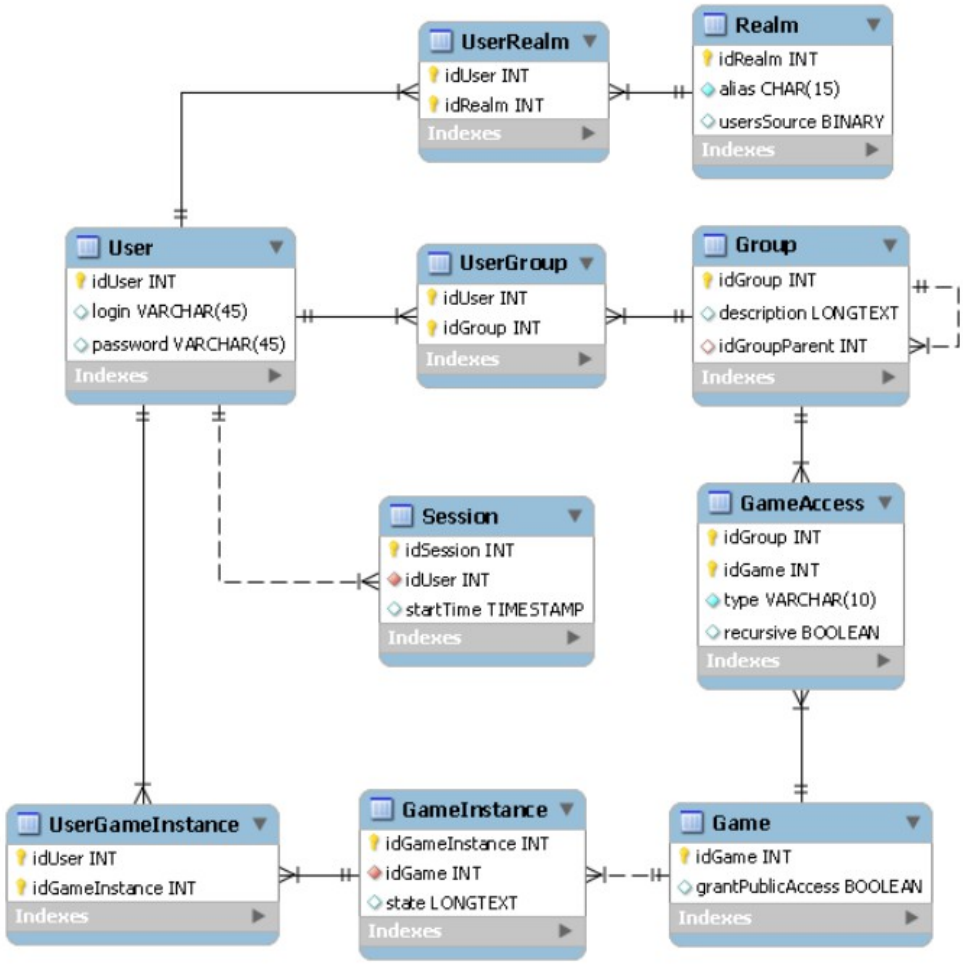
3. **GameLike** (se podría entender como Favorito): Representa un voto realizado por un usuario sobre un juego. Permite determinar si a un usuario le gusta un juego o no, resultando en algo así como un voto. Permite determinar la puntuación o popularidad de un juego entre los usuarios. Será utilizará para poder realizar una ordenación de una búsqueda por *los más populares*.

4. **Comment** (Comentario): Representa cada uno de los comentarios realizados sobre un juego. Esta clase será necesaria para poder realizar ordenaciones de búsquedas por los más comentados. Teóricamente elgg proporciona ya la gestión de comentarios.

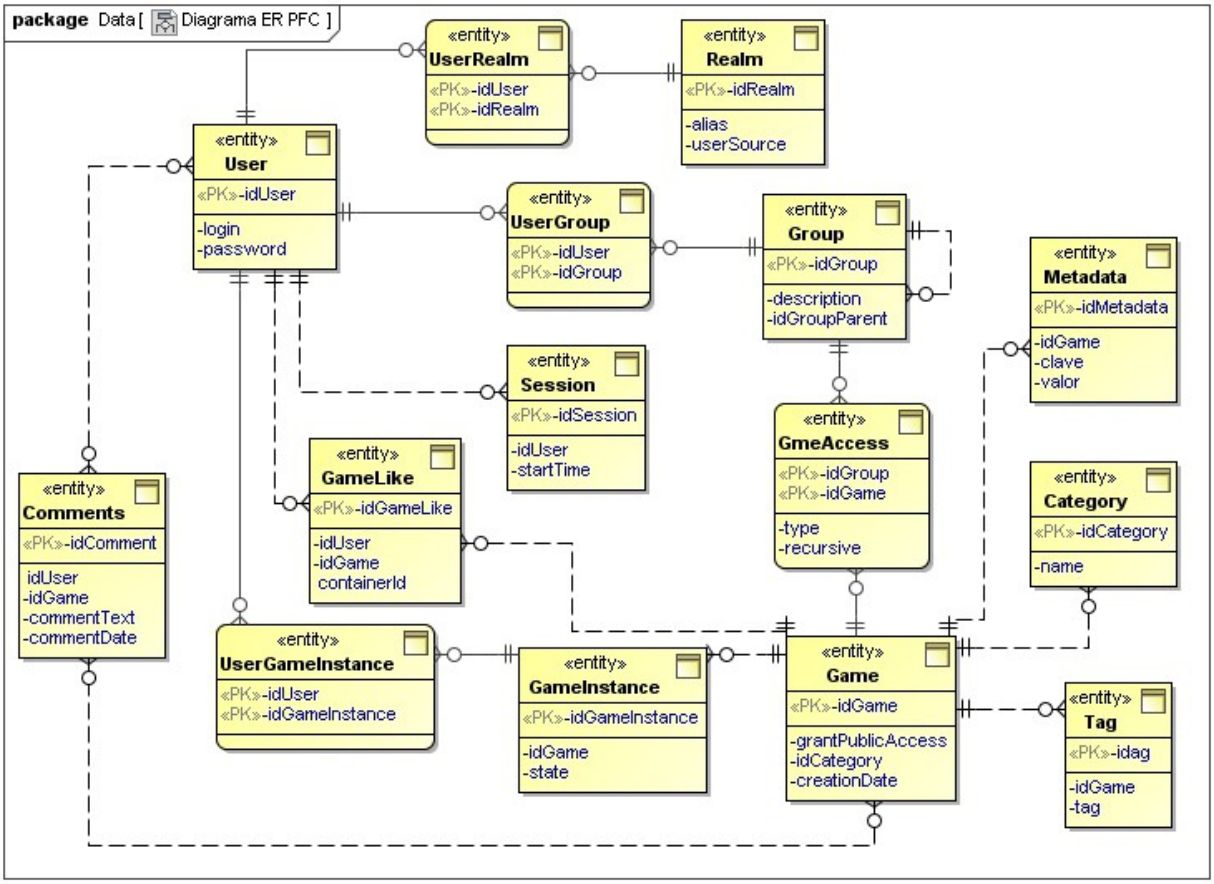
5. **GameInstance** (Instancia de juego) contiene información acerca de una instancia de un juego particular que debería contener información acerca del estado del juego. Permitirá realizar ordenaciones de búsquedas por los más activos, es decir, aquellos juegos para los que hay más jugadores jugando, actualmente.

6. **Category** (Categoría): Representa la categoría del juego. Deberá contener un campo que contenga el valor de la categoría. Permite determinar la categoría de un juego.

7. **Metadata** (Metadato): Contiene términos con sus valores que tratan de describir un juego.



La figura siguiente muestra la estructura de base de datos prevista según lo visto hasta el momento:



Prototipos de interfaz

La interfaz de usuario de una aplicación es a menudo el factor más importante para la aceptación del programa por parte de los clientes, en este caso, los usuarios del k-PAX. De modo que hay tener una cierta seguridad de que el programa que se va a desarrollar cumple con las expectativas del cliente.

Para crear el prototipo se han realizado técnicas de interacción humana-Ordenador. En concreto se han realizado las siguientes actividades que aseguran una mayor calidad del prototipo y en consecuencia de la aceptación del módulo desarrollado en este proyecto que tratará de asemejarse al prototipo:

1. Investigación con indagación contextual
2. Diseño de escenarios y flujo de interacción.
    1. Presentación de los escenarios.
    2. Diseño de flujos de interacción
3. Construcción de prototipado de alto nivel.
4. Evaluación del prototipo.

Por esta razón el proyectante ha creado un prototipo de la interfaz gráfica principal del proyecto. Observaciones:

1. Este prototipo ha sido creado con la ayuda de la herramienta *Justinmind* y los diagramas de flujos de interacción con *Magic Draw* y su Plugin en versión de prueba de 7 días *Cameo Data Modeler*. Esto reduce considerablemente el tiempo para implementar una interfaz gráfica de usuario (GUI).
2. Es una muy mala práctica utilizar un prototipo como parte del programa final de un proyecto, y crearlo con otro lenguaje evitará que el proyectante la utilice como la final.

Es seguro que el aspecto de la aplicación sufrirá cambios con respecto al prototipo, empezando por que no va a ser desarrollado con el mismo lenguaje, dado que el prototipo se hace con HTML puro y el programa del proyecto se hará con PHP, lenguaje con el que trabaja elgg, pero sí es cierto también que el aspecto general será lo más similar posible, de modo que se aprovechen las ventajas que su reflexión y análisis teórico proporcionan.

¿Quiénes son los usuarios? ¿Qué tareas tendrán que hacer? ¿Por qué utilizan el sistema? ¿Cómo utilizan el sistema? ¿Para qué utilizarán la aplicación? ¿Utilizan alguna aplicación similar? Si es así, ¿Qué les crea satisfacción y qué no les agrada? ¿Qué dispositivos utilizan?
Aunque estas preguntas son muy genéricas, sirven de punto de partida para determinar tres aspectos fundamentales:

- Conocer los usuarios.
- Conocer el contexto de utilización.
- Conocer las tareas que realizan.

Para conocer estos aspectos se ha aplicado la técnica de indagación contextual (*contextual enquiry*) que revela los detalles y las motivaciones implícitas al trabajo de las personas. El estudio, acerca al usuario al cual se observa y se pregunta, mientras realiza su trabajo o mientras interactúa con sus compañeros para acabar comprendiendo sus necesidades.

En este caso, se tendrán que observar usuarios navegando y consultando webs ya existentes, e identificar y proponer funcionalidades que se tendrían que integrar en la aplicación. Siempre teniendo en cuenta los límites establecidos en el apartado de *alcance*, en el *Documento de Objetivos del Proyecto*.

Aunque el proyecto ante pueda ser un usuario final de la aplicación, es importante recordar que no será el único usuario final del diseño. Por eso, las necesidades, objetivos y capacidades del proyecto antes no habrán de coincidir, necesariamente, con el resto de futuros usuarios.

Fase 1. Investigación.

Para la realización de esta fase he realizado 3 indagaciones contextuales entre amigos y familia, aparte del mío propio. La naturaleza innovadora de la plataforma dificulta en gran medida encontrar plataformas de redes sociales con la capacidad de jugar juegos en la misma plataforma. La información obtenida de estas encuestas se recoge a continuación. Las encuestas se han realizado sobre portales de redes sociales que proporcionan la posibilidad de jugar varios juegos sociales, tales como facebook, tuenti, twitter, etc.

Definición de los perfiles de usuario

Se pueden observar diferentes perfiles de usuarios:

- Personas sin experiencia alguna en el manejo con redes sociales.
- Personas sin experiencia alguna en realización de redes sociales, pero sí en búsquedas de productos.
- Personas que han utilizado redes sociales anteriormente, pero con la ayuda de otra persona más experimentada. (Haciéndolo casi todo esa otra persona)
- Personas poco experimentadas pero que han utilizado redes sociales anteriormente
- Personas experimentadas que han utilizado redes sociales muchas veces.
- Personas discapacitadas en cualquiera de las situaciones anteriores.

Definición del contexto de uso de cada usuario que se observa durante la observación contextual

Como ya he indicado anteriormente, se han realizado 3 indagaciones contextuales.
En cuarto lugar me incluyo a mi mismo como un usuario del perfil de los que han utilizado poco las redes sociales.

Las razones para realizar búsquedas de videojuegos de cada persona son diferentes. Así que pondré cada situación en su contexto:

- El primer usuario, se trata de una persona que ha realizado búsquedas anteriormente en buscadores de juegos y que conoce medianamente bien el funcionamiento de las redes sociales. Está aburrida y quiere encontrar un juego de carreras para divertirse un rato en su red social favorita, Facebook. No desea sin embargo pagar por jugar el rato que vaya a estar. Su objetivo es por tanto, encontrar un juego de carreras gratuito desde el portal de Facebook.
- El segundo usuario, es una persona que nunca ha utilizado redes sociales ni buscado juegos en portales de mini juegos. Quiere encontrar un juego de lógica tipo sudoku sin necesidad de descargarse el juego, aunque no le importa si es necesario.
- El tercer usuario, es una persona que ha realizado muchas veces búsquedas en portales de mini juegos pero que apenas sí ha utilizado portales de redes sociales. Alguien le ha recomendado un juego del portal de tuenti y quiere probarlo. Su objetivo es encontrar y jugar al dicho juego.

Definición en detalle de las tareas que realiza cada usuario observado.

1. ***Observación:***
   En el primer caso de observación, la usuaria se encuentra navegando por su red social favorita, Facebook. Ha terminado de ver fotos comentarlas con sus amigas y leer sus

mensajes y ahora quiere jugar un juego de carreras. En esto que se le ocurre que puede buscar en el buscador de Facebook algún juego de este tipo.

Para ello, pulsa sobre el buscador y escribe el literal “carreras”. Tras esto, el servidor le devuelve un listado de todas las personas que se describen con dicha palabra. Esto molesta a la usuaria por no tener un buscador aparte para los juegos de Facebook, sin embargo, no deja de internarlo.

Una segunda vez, trata de buscar juegos, esta vez, con el literal “juegos de carreras”. En esta ocasión, el servidor devuelve un listado flotante de lo que parecen ser juegos, con su nombre y una imagen.

La usuaria escoge el que más le atrae y hace clic sobre su nombre. Como respuesta, el servidor muestra la página de facebook de lo que parece ser alguna empresa desarrolladora de juegos flash. En esta se muestra el logotipo del juego, su nombre, una descripción y un vínculo para jugar al juego.

La usuaria, animada, hace clic sobre el vínculo para jugar al juego, pero para su sorpresa, Facebook requiere el registro previo de la usuario mostrando un formulario donde debe introducir ciertos datos personales. Esto desagrada mucho a la usuaria que de ningún modo desea entregar sus datos para jugar un juego. La usuaria se molesta tanto que cierra el formulario y decide no buscar más juegos de esta manera.

2. ***Observación:***

En el segundo caso, lo primero que hace la usuaria es acceder al portal. En ella aparece un listado de categorías de juegos entre los que busca la palabra sudoku o similares. Al no encontrarlo, se decide por una categoría concreta, tras lo que aparece el listado de juegos de dicha categoría. Esta lista no se encuentra ordenada alfabéticamente y aparentemente el sistema no permite ordenar la lista así. Como consecuencia, la usuaria se irrita.

Ante esta situación se decide por buscar el juego en algún buscador de la página, pero no lo encuentra. Lo cierto es que sí que lo hay pero no se destaca visualmente por estar pintado con un color similar al del fondo del portal. Eso le lleva a confundirse y acceder a una página ajena similar. Poco tiempo después se da cuenta de su error, aunque prefiere continuar en este nuevo portal.

En este segundo portal sí hay un motor de búsqueda en la que la usuaria realiza una búsqueda de juegos por medio del literal “sudoku”. La página, tras realizar supuestamente la búsqueda, devuelve una página con muchas descripciones de juegos. Tras ello, la usuaria se molesta porque la página no sólo no le indica de ningún modo que no ha encontrado juegos cuya descripción contenga la palabra sudoku sino que además los juegos que le muestra no se acercan siquiera a las categorías con las que se podrían englobar la familia de juegos de los sudokus. Dándose por vencido, vuelve al portal inicial.

Buscando de forma más detenida se da cuenta de que efectivamente sí existe un motor de búsqueda en el que se muestra el literal “Qué buscar”. La usuaria hace clic sobre el campo de filtrado del motor de búsqueda, pero el literal no desaparece. Esto confunde a la usuaria que piensa que puede ser que en realidad no se trate de un motor de búsqueda. Para asegurarse trata de borrar el literal y entonces sí, escribe el literal “sudoku” para pulsar inmediatamente después en el botón de buscar.

Como resultado la página le devuelve un listado flotante con los juegos encontrados.

La usuaria se alegra porque por fin ha encontrado lo que buscaba. El listado le ha devuelto un conjunto de juegos de sudoku con imágenes y descripciones de cada juego, así como su nombre.
Una vez escogido uno, la usuaria hace clic sobre este y la página le devuelve una página en la que inmediatamente se muestra el juego cargándose. Finalmente la usuaria ha conseguido su objetivo y disfruta del juego durante su tiempo libre.

3. ***Observación:***

En el tercer caso, hablando con un conocido un juego RPG recientemente salido a la venta, el conocido comenta al usuario que en Facebook se puede jugar un juego de forma gratuita y muy similar, su nombre "Dragon Rampage".

El usuario por razones personales aborrece Facebook y en especial las redes sociales, por lo que no tiene especial experiencia con este tipo de portales. Sin embargo recuerda que tiene una cuenta en twitter desde hace un tiempo, aunque no la ha utilizado prácticamente. Piensa que quizás pueda jugar también al juego si lo hace desde twitter.

Para ello accede primero a su cuenta de twitter. El sistema le da la bienvenida y muestra entre otras cosas el motor de búsqueda de twitteros. Puesto que ha pasado un tiempo desde que habló con su conocido, y dado que tiene mala memoria, el usuario ya no recuerda el nombre del juego, de modo que se decide por buscar el literal "juegos rpg" para tratar de encontrarlo en una hipotética lista de juegos de la categoría de juegos rpg.
Al pulsar sobre el botón de buscar, el sistema le devuelve un listado de todos los twitteros que han hecho comentarios con todo o parte del literal buscado. Naturalmente esto fastidia al usuario que se esperaba le mostrara un listado de todos los juegos de la categoría rpg que estuvieran adheridos al portal.

Sin rendirse, trata de buscar el juego esta vez por medio de literal "juegos". Una vez más al pulsar sobre el botón de buscar, el sistema devuelve un listado de personas que comentan juegos, lo que enfada aun más al usuario que se ve obligado a llamar a su conocido para preguntarle por el nombre exacto del juego, y satisfacer así su deseo de jugar al juego.

Una vez recordado el nombre, lo escribe en el motor de búsqueda por medio del literal "Dragon Rampage". En esta ocasión, al pulsar sobre el botón de buscar, el sistema le devuelve un listado de tweets entre los que aparece uno con el nombre "Dragon Rampage" y un icono con su logotipo.

El usuario se relaja pensando que por fin ha encontrado el juego que quería. Pero al pulsar sobre el tweet, el sistema no le devuelve a ninguna página donde pueda jugar directamente al juego, ni siquiera a una descripción del mismo, sino que le muestra un listado de los tweets que la compañía del juego ha realizado en su cuenta de twitter y que están visibles al usuario. Debido al poco conocimiento del usuario sobre el funcionamiento de la red social, le desagrada especialmente.

Mirando más detenidamente en esta lista, ve un link a la página de Facebook del juego. Esto irrita si cabe aun más al usuario que entiende que no puede jugar desde twitter y sí desde Facebook a la que por principios y razones personales no va a acceder. Finalmente abandona twitter una vez más con una mala experiencia.

Fase 2. Diseño de escenarios y flujo de interacción.

A partir del contexto de uso, hace falta generar los escenarios de uso. Los escenarios son una herramienta que facilita hacer hipótesis sobre las situaciones en les que se encontraran los usuarios y las necesidades que tendrán para llevar a cabo sus objetivos.

Por lo tanto, no son una manera de documentar las interacciones concretas de un usuario, sino de poner de manifiesto el contexto de uso y los objetivos de los usuarios con sus motivaciones. A partir de estos escenarios se puede generar una estructura de la aplicación y los flujos de interacción que se producen.

Estos flujos han sido generados gráficamente con la herramienta de prototipado Magic Draw, utilizando el vocabulario visual, para describir la información y el diseño de interacción, de Jesse James Garrett que es posible encontrar en http://www.jjg.net/ia/visvocab/spanish.html, donde se encuentra también una biblioteca de formas utilizables con el Microsoft Visio, aunque esta biblioteca no ha sido necesaria porque Magic Draw contiene la suya propia.

Presentación de los escenarios

Para la creación de los escenarios se han tenido en cuenta los principales problemas que se encontraron los usuarios durante las indagaciones contextuales, así como lo que les agradó.

Escenario 1:

Es un domingo por la tarde y Jesús hoy no trabaja, de modo que invierte su tiempo libre en sacarse la carrera universitaria de la UOC. En estos momentos ha terminado su última PEC y piensa que se ha merecido un descanso antes de comenzar con la siguiente. Sin embargo le gustaría utilizar su tiempo para mejorar intelectualmente. Para ello accede a k-PAX en busca de algún juego corto que le llene el pequeño espacio de tiempo que tiene de forma que le sea de utilidad.

No conoce los nuevos juegos que se han introducido en k-PAX así que accede al listado de todos los juegos, para ordenarlos según su antigüedad e ir mirando la descripción de los juegos para hacerse una idea de lo que ofrece k-PAX, actualmente.

A medida que ve diferentes juegos, se interesa en una categoría de juegos concreta, de modo que decide filtrar los juegos por dicha categoría y buscar en el listado de juegos el que más le llama la atención visualizando sus nombres, descripciones, logotipos y demás campos genéricos.

Mientras se decide, con un solo clic, selecciona aquellos juegos que más le gustan para que se muestre su ficha y los comentarios de los demás jugadores.

Cuando se decide, accede a la ficha del juego concreto, desde donde comienza una partida nueva. Como es usuario de la plataforma k-PAX y ya ha accedido al mismo, no le hace falta registrarse de nuevo como usuario ni tampoco como nuevo jugador. El sistema le identifica automáticamente.

Una vez terminada la partida, el usuario se decide a marcar positivamente el juego, puesto que le ha gustado y piensa que a otros les podría interesar. Además también se decide a dejar un comentario para que otros conozcan su opinión.

Finalmente, Jesús mira la hora y se da cuenta de que debería seguir. Cierra la sesión y comienza a realizar su siguiente PEC, más relajado y sintiéndose un poquito más sabio, gracias a lo que ha aprendido en el juego.

Escenario 2:

María se encuentra leyendo los comentarios de sus amigos en su red social favorita, donde descubre por comentario de uno de ellos la existencia de una plataforma de red social que además de las actividades propias de las redes sociales, ofrece multitud de juegos educativos. Esta idea, a María, le gusta, recordándose a sí misma que recientemente quiso buscar en su red social un juego en el que pasar el rato. Si esta plataforma puede solucionar su problema y además mejorar sus capacidades cognitivas, piensa, mucho mejor. De modo que decide darle una oportunidad.

Ella accede a la página que su amigo le dice, donde advierte que debe introducir sus credenciales para acceder o por lo contrario registrarse primero. Deduce entonces que para jugar juegos es necesario primero hacerse miembro de la plataforma.

Realiza exitosamente el proceso de registro, y accede por primera vez al portal. Una vez que ha iniciado una sesión, hecha un vistazo a la interfaz y se da cuenta de la presencia de dos elementos. Un buscador y una pestaña dedicada a juegos. Como lo que quiere hacer, imperiosamente, es jugar juegos, se decide por entrar en la sección de juegos.

El sistema le muestra el listado de todos los juegos asociados a la plataforma de forma paginada; mostrando su logotipo, nombre, una pequeña descripción, un conjunto de tags que clasifican al juego según diferentes términos y su puntuación; y permitiendo la ordenación por diferentes criterios de búsqueda así como el filtrado de la búsqueda por otro conjunto de criterios.

No está segura de a qué jugar de modo que decide ordenar los juegos según los más valorados. El sistema inmediatamente reordena la lista de juegos según la puntuación de los juegos. Echando un vistazo a la descripción de los juegos se decide por uno y lo selecciona pulsando sobre éste mostrándose la ficha del juego, con su logotipo, nombre, descripción, tags, comentarios, puntuación, etc. y, por supuesto, la posibilidad de comenzar una partida.

María lee la descripción del juego y los primeros comentarios de los jugadores y sin dudarlo comienza una nueva partida. Al terminar, María está encantada y decide jugar un juego similar, de modo que busca entre los tags de la ficha del juego aquel que mejor define el término del tipo de juego que desea y lo pulsa. El sistema le devuelve al listado de juegos, pero esta vez filtrándolos por el tag seleccionado.

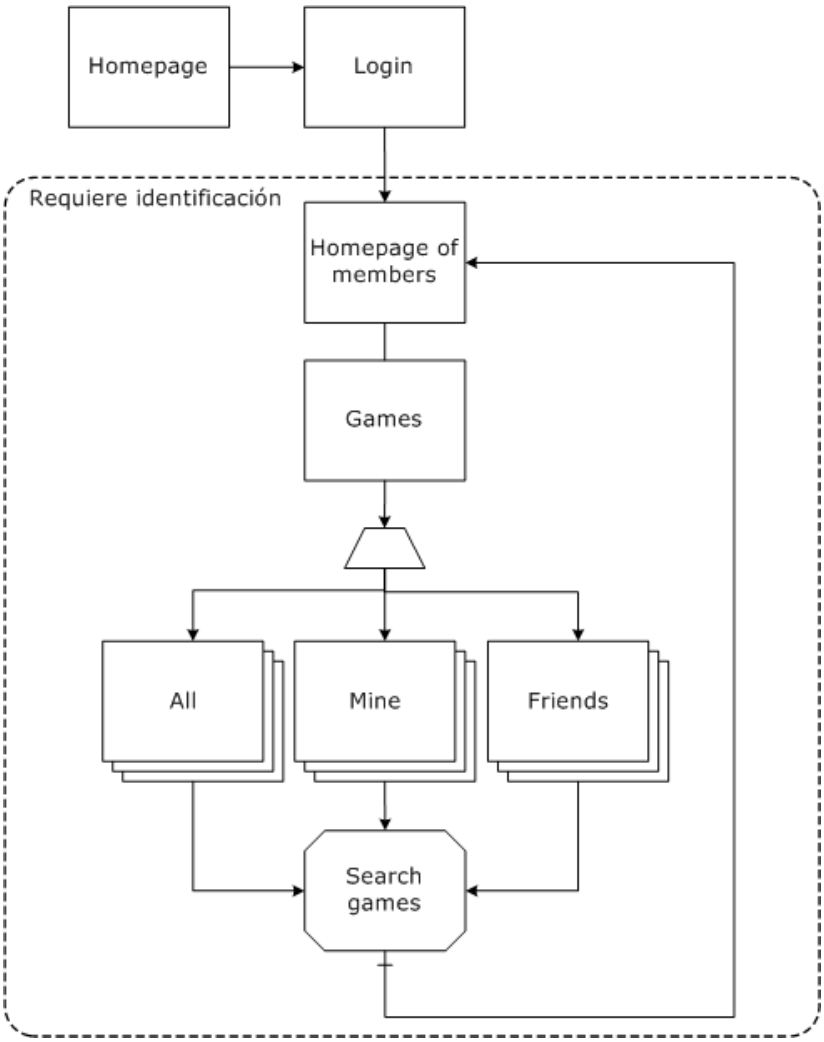
Entre el listado de juegos mostrados encuentra otro que posiblemente le guste. Accede a su ficha para comenzar una nueva partida. Al finalizar decide que le ha gustado más el primero, de modo que decide puntuarlo. Puntuar el juego en el que se encuentra actualmente es trivial, puesto que se muestra la posibilidad de votar el juego en la ficha de cada juego e incluso en el listado.

Para votar el otro juego ha de volver a la ficha del mismo. Para ello accede al listado de juegos y filtra los juegos indicando el nombre. Sólo se acuerda de una palabra del nombre del juego, de modo que lo introduce en el filtro y el sistema vuelve a listar los juegos que se nombran conteniendo dicha palabra. Entre los juegos listados aparece el que María estaba buscando. Tras lo cual ejerce su voto.

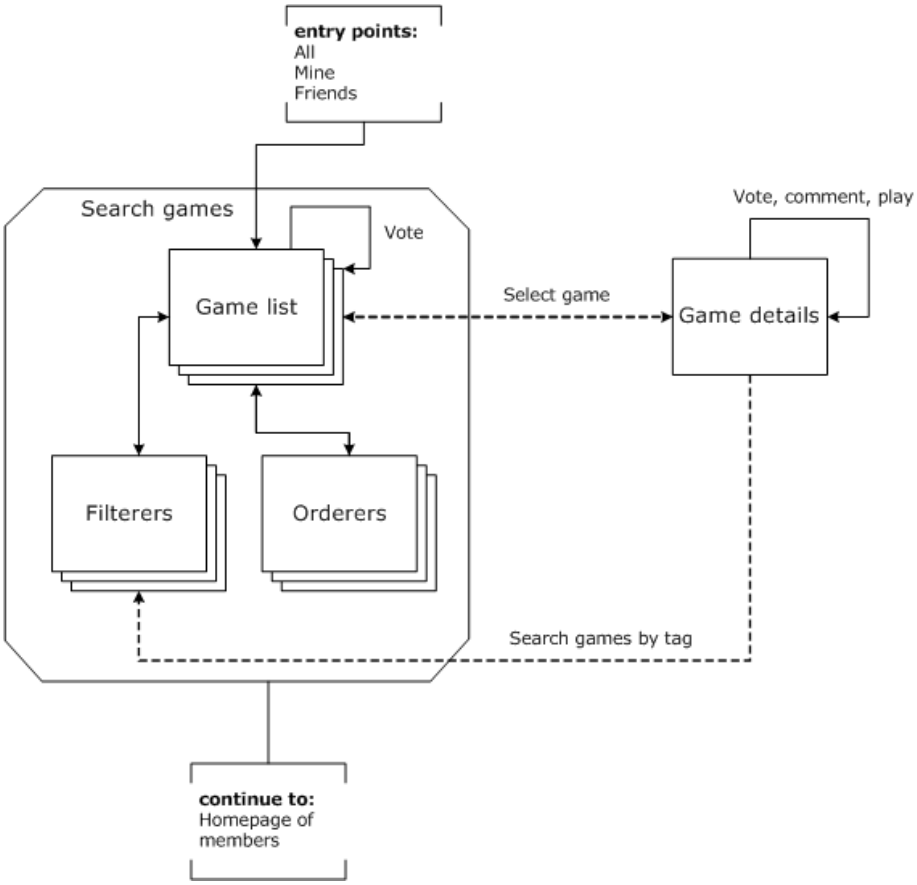
Finalmente abandona la plataforma para dejarle un comentario a su amigo en su red social habitual agradeciéndole el consejo de registrarse en k-PAX.

Flujos de interacción

A partir de los escenarios anteriores, pueden adivinarse las necesidades de flujo de cada página del portal obteniéndose así los diagramas de flujo de interacción siguientes:[1]



1 - En la página de login el usuario se identificara y autenticará con sus credenciales pasando a la pantalla principal de miembros si lo hace correctamente o mostrándose un error.



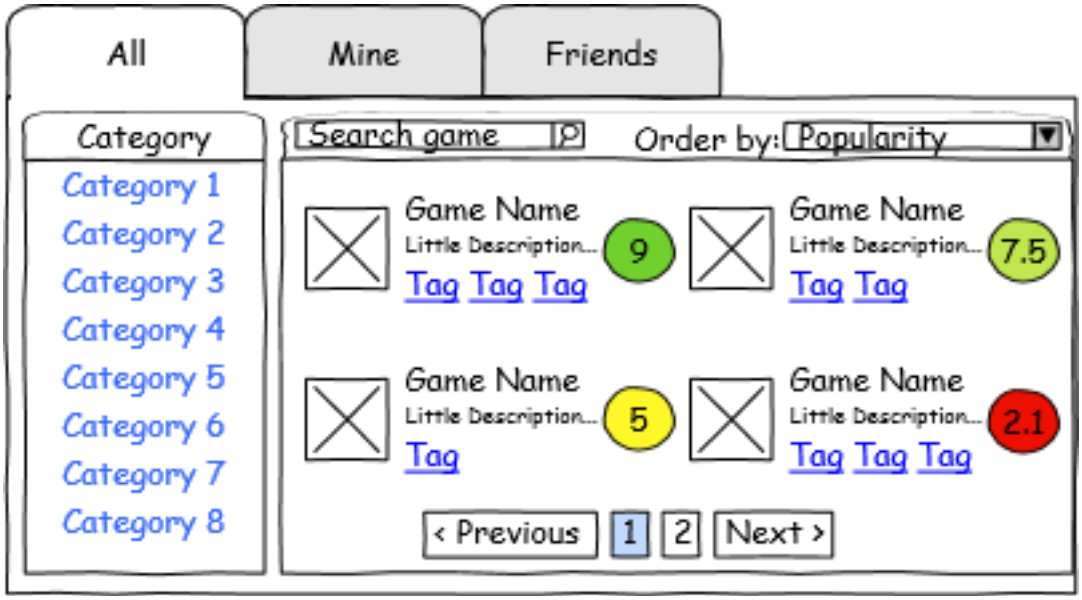
Fase 3. Prototipado.

He realizado el diseño conceptual del nuevo módulo a implementar según la información obtenida en las fases anteriores. Las imágenes que se muestran en este apartado representan las pantallas de listado de juegos y de ficha de un juego, donde se podrán realizar las funcionalidades de búsqueda de juegos y consulta de los detalles de un juego concreto, respectivamente.

Para ello he utilizado el programa Pencil, de código libre y disponible gratuitamente desde su página oficial. Los dibujos realizados con este programa son un tanto sencillos ya que parecen realizados a mano alzada, sin embargo representan perfectamente las necesidades de la interfaz de usuario.

Los dibujos conceptuales aquí descritos no son de ninguna manera definitivos puesto que se su situación, tamaño y forma se verán afectados por cuestiones de diseño e implementación. No obstante representa a grandes rasgos el aspecto final del módulo.

En la figura siguiente se puede ver el diseño conceptual de la pantalla de búsqueda de juegos:



La pantalla de búsqueda de juegos formará parte de la pestaña de juegos (games), existente actualmente en la cabecera de la página principal de k-PAX. Dicha pestaña contiene actualmente otras tres pestañas: All, Mine, Friends. Estas pestañas y su contenido son las que se representan en este dibujo.

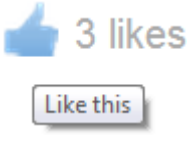
El contenido de cada pestaña será el mismo, con la diferencia de que las funcionalidades de búsqueda, filtrado y ordenación se aplicarán a un conjunto diferente de juegos: todos los juegos, los del propio usuario o los de los amigos del usuario, respectivamente, para las pestañas de All, Mine y Friends.

La pantalla se divide verticalmente por dos elementos. El primero, situado en la parte izquierda, es una lista de todas las categorías de todos los juegos disponibles en la plataforma. Seleccionando uno de ellos, se realizará el filtrado por categoría, mostrando el listado de juegos de dicha categoría para el conjunto de juegos que corresponda (todos, los del usuario o los de los amigos del usuario) y por supuesto aplicando el resto de filtros y modo de ordenación escogidos.

En la parte derecha se muestra principalmente el listado de juegos encontrados que cumplan las condiciones de la búsqueda. De arriba a abajo y de izquierda a derecha se muestran los siguientes elementos:

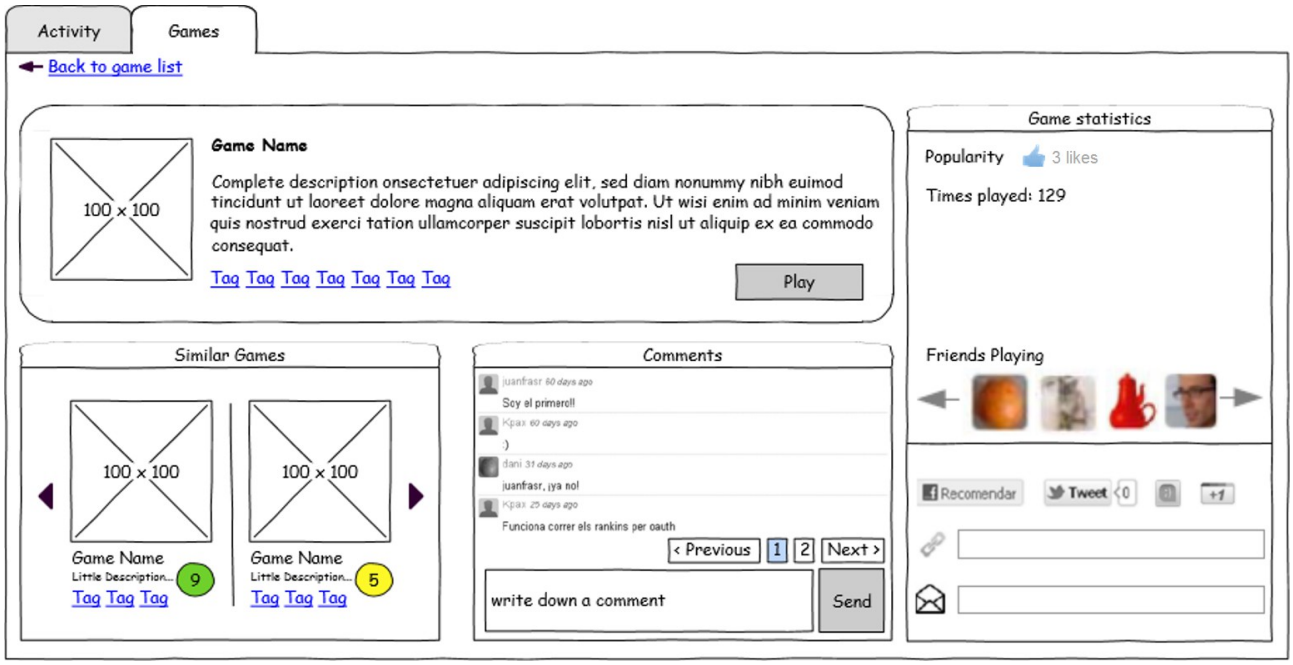
- Un campo input para realizar la búsqueda de juegos por nombre
- Un combo que permitirá escoger entre todos los métodos de ordenación. Sólo podrá haber un método de ordenación seleccionado en cada momento
- El listado de juegos, donde para cada uno se mostrará una imagen o logotipo, el nombre del juego, una descripción breve, correspondiente a las primeras palabras de la descripción completa del juego -dependiendo del espacio que ocupe en general cada juego, es posible que este campo no se termine introduciendo, puesto que será accesible desde la ficha del juego y puede no dar suficiente información si se trunca.-
- La popularidad del juego.
- Los elementos que permiten la paginación del listado.

La popularidad del juego será representada como hasta ahora, con una mano con el dedo pulgar apuntando hacia arriba. Se pintará azul si se quiere marcar o se ha marcado como que al usuario le gusta o en gris sin no ha sido votada como tal. Sin embargo en el dibujo se ha representado con círculos de colores por ser más rápido



de dibujar. En la figura de la derecha podrá sin embargo ver un ejemplo de como se visualizaría la popularidad de un juego ya que está obtenido de la página de kPAX.

La siguiente imagen muestra el diseño conceptual de la ficha de un juego. Este diseño ha sido adaptado a partir de la idea de diseño que se encontraba en la documentación en inglés de la plataforma, añadiendo o modificando los elementos según la información obtenida en las dos fases anteriores. Este dibujo se ha escogido por considerarlo suficientemente bien diseñado, siguiendo los conceptos fundamentales de la Interacción Persona Ordenador (IPO).



En la figura, puede encontrar de izquierda a derecha y de arriba a abajo, los elementos siguientes:

- En el bloque superior izquierdo, los datos básicos del juego, como son, nombre, descripción completa, Tags entre otros. Además es aquí donde se podrá solicitar comenzar una partida al juego por medio de un botón “Play”. En caso de realizarse la edición de metadatos, se mostrarían entre la descripción y los tags.
- A la derecha en un bloque vertical que ocupe el alto de la pantalla, los datos estadísticos del juego tales como su popularidad, junto con la posibilidad de indicar si le gusta el juego o no, el número de veces jugado y el número total de usuarios a los que les gusta el juego. Se mantiene la idea de mostrar quiénes están jugando en estos momentos (que no se hará en este proyecto).
  La parte inferior del bloque se reserva vacía, por ejemplo para añadir elementos que permitan compartir el juego en otras redes sociales, tales como Facebook, twitter, google+... también la posibilidad de enviarlo por correo, y más. Esta parte, sin embargo, tampoco se implementará por estar fuera de los límites del proyecto.
- En el bloque de la esquina inferior izquierda un listado de juegos similares al actual, teniendo en cuenta que sean de la misma categoría o que contengan al menos un mismo tag. Cuantas más características tengan en común aparecerán con mayor anterioridad en el listado. Para poder navegar entre los juegos del listado, se podrá paginar por medio de las flechas situadas a la derecha e izquierda del bloque.
- En el bloque inferior central, se muestra el listado de comentarios realizados por los usuarios de k-PAX con respecto al juego. También permite la paginación e insertar un nuevo comentario

Finalmente, como resumen, el diseño de la interfaz del usuario estará bien cuidada con el uso de

colores de fondo claros y de texto oscuros con pocos tonos diferentes y perfectamente combinados para ser legibles junto con el uso de fuentes de letras no demasiado ornamentales, siguiendo la codificación de cada color para la cultura española, con el uso de iconos más simples cuanto más pequeños sean, uso de metáforas y estándares para su buena interpretación, manteniendo la ubicación de los elementos que se repitan en el mismo sitio, con el uso de imágenes o tamaños de letra mayores para los elementos más usados o de mayor importancia, para mantener su jerarquía visual, y utilizando las leyes de Geslalt para agrupar cada elemento según su relación con el resto.

Como aclaración, ya se ha indicado que la puntuación de un juego será visualizada como hasta ahora con manos con pulgares apuntando hacia arriba y no como un sistema de colores. No obstante, se ha mostrado así en el diseño conceptual por las limitaciones del programa con que se realizaron. Igualmente las letras pequeñas de las figuras anteriores son limitación del programa de diseño de interfaz de usuario y en la realidad se buscará que sean lo suficientemente grandes y escalables.

Posibles mejoras

Durante el desarrollo del proyecto ha habido requisitos que no han sido solicitados al proyectante. Sin embargo, se han pensado en ellos como una posible mejora del sistema.

A continuación, se definen las posibles mejoras del sistema. Algunas de ellas ya se han indicado anteriormente

- Se propone utilizar la parte inferior del bloque de la derecha en la ficha de un juego para añadir elementos que permitan compartir el juego en otras redes sociales, tales como Facebook, twitter, google+... también la posibilidad de enviarlo por correo, añadirlo con un link en un blog, etc.
- Al buscar un juego por medio del motor de búsqueda de elgg, ya sea por la categoría, tag o algún elemento que los describa, se muestre el listado de juegos que atienda al término o términos buscados.
- En caso de no implementarse, la posibilidad de añadir tantos metadatos como se desee.
- Al visualizar la ficha de un juego, mostrar los usuarios que están jugando en ese momento.
- Poder añadir y eliminar categorías por medio de un formulario y no exclusivamente desde base de datos.

Especificación de plan de pruebas

Se va a establecer un plan de pruebas que permita comprobar si el sistema satisface o no los requisitos planteados anteriormente.

Plan de pruebas:

1. **Pruebas unitarias**: Se especifican en cada iteración del proyecto y se realizan al final de cada una. Su objetivo es comprobar que el entregable cumple con los requisitos.

   Las pruebas serán de caja negra centradas para comprobar la correcta funcionalidad del sistema, y de caja blanca durante la codificación de las aplicaciones.

2. **Pruebas de integración**: Se ejecutarán tras las pruebas unitarias, su finalidad es comprobar que cada incremento añadido al sistema se integra y funciona correctamente con el incremento anterior.

3. **Pruebas de sistema**: Estas pruebas se realizarán cuando se hayan terminado todos los incrementos, con sus respectivas pruebas y se tenga el producto completo.

Su objetivo es comprobar que el sistema completo funciona correctamente y cumple todos los requisitos.

Se tendrán en cuenta diferentes factores como el rendimiento.

4. **Pruebas de aceptación**: La realiza el cliente. Su objetivo es comprobar la satisfacción del cliente con el sistema.

Diseño

Introducción

En la fase de análisis se han establecido los requisitos del sistema, casos de uso y en general qué debe hacer el nuevo módulo que se implante en k-PAX.

Como ya se ha dicho, el nuevo sistema debe utilizar un Sistema Gestor de Bases de Datos (SGBD) y debe mostrar los datos por medio de la red social elgg. Además el intercambio de información entre elgg y la parte base de datos se realiza por medio de servicios web implementados en Java.

En la fase actual de diseño se definirán las clases del sistema a partir de la estructura definida en la fase de análisis para un modelo dividido en capas, así como las interfaces de cada capa, todo ello cumpliendo con los requisitos del sistema y garantizando un mínimo de seguridad, calidad y rendimiento.

En esta fase, también se diseñará la estructura de la BD para que se amolde a los requisitos del nuevo módulo. Así mismo, la BD deberá ser diseñada de modo que se puedan hacer consultas y modificaciones a ella de forma concurrente desde diferentes lugares y garantizar su integridad con la realidad que modele.

Por último en esta fase, se definirán las pruebas de integridad que deba cumplir el sistema.

Descomposición en capas

Actualmente, la plataforma k-PAX está dividida en varias capas organiza el sistema haciendo que cada elemento trabaje a un único nivel de abstracción por medio del patrón de diseño Arquitectura en Capas, estableciendo 3 capas, las comunes, presentación, lógica de negocio y persistencia, donde la capa de presentación y lógica de negocio utilizan además el patrón de diseño MVC (modelo-visa-controlador), para organizar las responsabilidades de interacción con el usuario.

Además, para ofrecer un único punto de entrada al subsistema, la capa de lógica está diseñada por medio del patrón de diseño fachada, que se implementa con servicios web. Este a su vez utiliza la capa de persistencia por medio de otro patrón “Fachada”, implementado con el ORM Hibernate para poder acceder a un servidor de base de datos sin casar el código con el SGBD.

Este diseño de patrones deberá ser mantenido por el nuevo módulo al término del proyecto.

Definición de capa de persistencia

Como ya se ha dicho en la fase de análisis la aplicación resultante del proyecto utilizará una BD para almacenar los datos necesarios bajo el SGBD MySQL en su versión 5.

La capa de persistencia del proyecto debe ser capaz de añadir, modificar, borrar, y consultar datos de la BD, así como de mantener la configuración deseada para la aplicación en cada ordenador en el que se ejecute.

Definición de capa de lógica de negocio

Deberá diseñarse una capa que tome los datos necesarios para la realización de los casos de uso

definidos en la fase de análisis y que compruebe la validez de los mismos, es decir, que todo aquello que implemente dicha capa deberá estar implementando en la lógica de negocio del sistema y por tanto formar parte de ella.

La capa de lógica del sistema resultante del proyecto se encargará de validar los datos introducidos por el usuario, comprobar que el usuario tiene permiso para realizar las acciones solicitadas, realizar cálculos, comunicarse con servicios externos y de servir de puente entre la capa de presentación y el mecanismo de persistencia, así como de comunicar errores internos a la capa de presentación.
Su interfaz deberá indicar las es necesarias para la realización de las operaciones da lógica de negocio del sistema.

Definición de capa de presentación

La capa de presentación es una capa arquitectónica que se encarga de proveer una interfaz gráfica de usuario que presente los datos al usuario, es decir, que la capa de presentación que se diseñe para el sistema del proyecto deberá servir de interfaz entre la capa de negocio y el usuario que lo utilice.

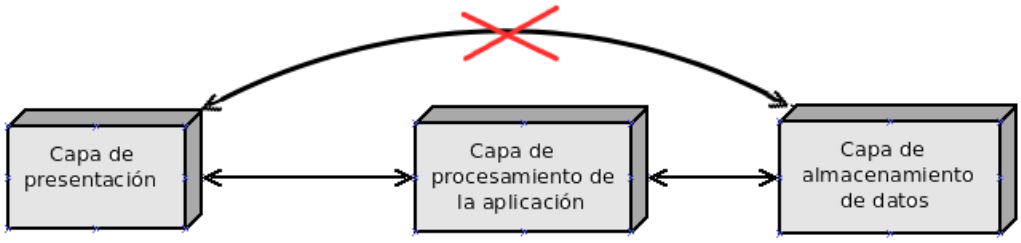
El prototipo realizado durante la fase de análisis sirve como herramienta para conocer los gustos del usuario y permitirá al proyectante diseñar una capa de presentación que se le adapte lo mejor posible para consiguiendo que ésta sea simple intuitiva y agradable.

Relaciones entre capas

Las capas se relacionan entre sí en forma de canal. Una capa cualquiera sólo puede relacionarse con su inmediatamente anterior o siguiente.

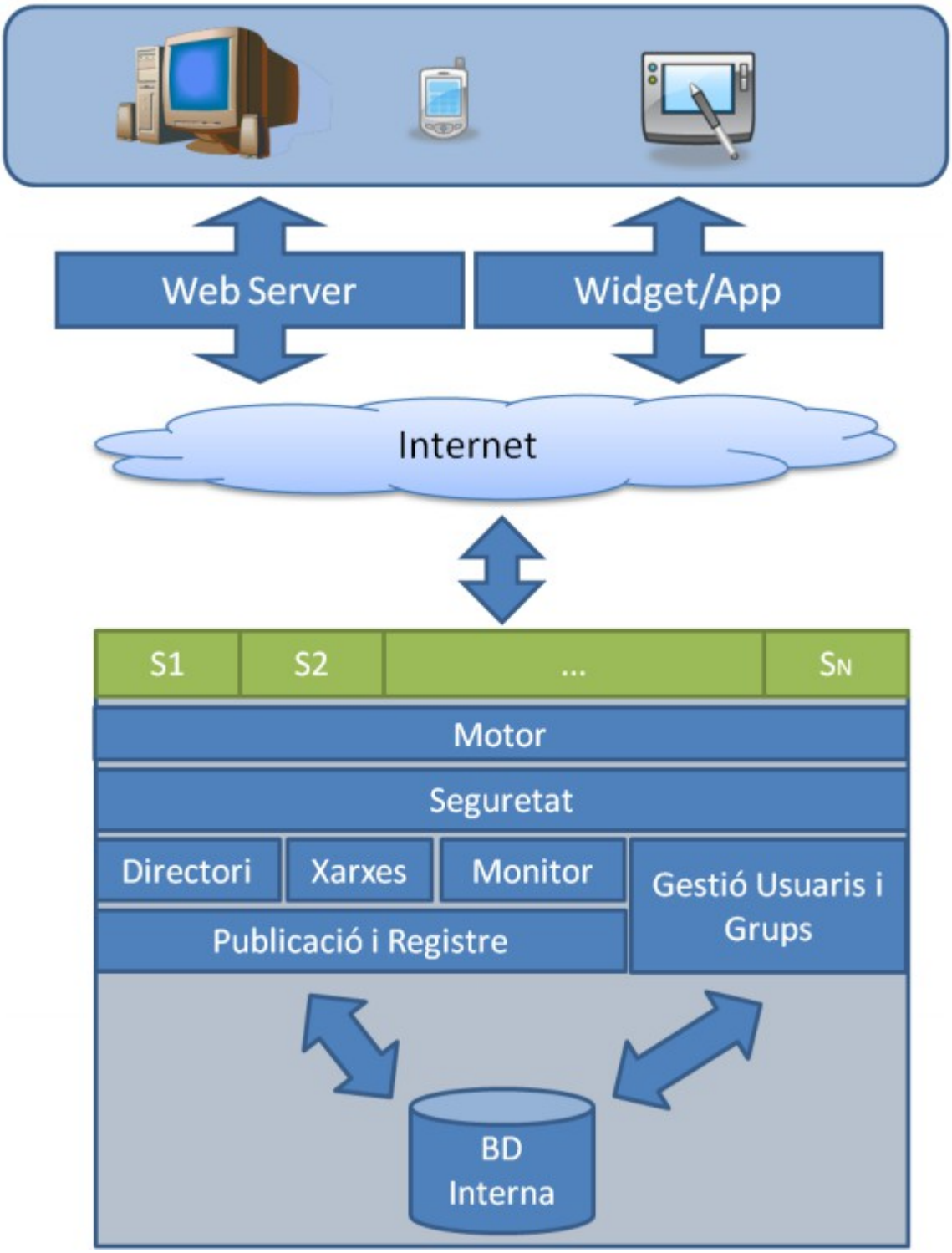
En el caso de este proyecto, al definirse 3 capas, la capa de presentación y persistencia sólo pueden relacionarse con la de lógica y ésta con ambas. Nunca una capa puede relacionarse con otra saltándose capas intermedias, es decir, que en el caso de este proyecto, la capa de presentación y la de persistencia nunca han de poder relacionarse entre sí, si no es mediante la capa de lógica.

A continuación, se muestra una figura que representa las relaciones entre las capas de persistencia, lógica y presentación:



Descomposición física

El sistema actual que está implantado en la plataforma k-PAX se descompone físicamente en diferentes elementos. En la siguiente figura, obtenida del documento técnico en inglés que describe la plataforma k-PAX, se muestra esta descomposición:



El nuevo sistema se descompondrá físicamente de la siguiente manera:

1. Capa de persistencia:

a. SGBD.
b. BD de la aplicación.
c. Interfaces DAO y sus implementaciones que permiten obtener o modificar los datos almacenados independientemente de como se almacenen o de dónde se haga. Sirven de canal entre la BD y la capa de lógica. Están almacenadas en el paquete uoc.edu.svrKpax.dao.
d. Clases de intercambio de información, que sirven para transferir los datos de una entidad entre las distintas capas. Almacenados en el paquete uoc.edu.srvKpax.vo
e. En su caso, ficheros de configuración, incluidos en la carpeta main/resources.

2. Capa de lógica:
   a. Interfaces BO y sus implementaciones que hacen de canal entre la capa de persistencia y la de presentación. Están albergadas en el paquete uoc.edu.srvKpax.bussines.
   b. Servicios web o funciones ofrecidas para que sean accedidas desde la capa de presentación con elgg. Incluidas en el paquete uoc.edu.srvKpax.rest
   c. Resto de clases de soporte para los módulos de Security, Authorization, etc. Incluidas en el paquete uoc.edu.srvKpax.util.
3. Capa de presentación:
   a. Elgg.
   b. Clases y ficheros PHP que hacen de canal entre el usuario y la capa de lógica y que implementan la GUI de la aplicación por medio de llamadas a los servicios web.

Librerías

Para la elaboración de la aplicación el proyectante se ayudará de un conjunto de librerías que agilicen el proceso de implementación de la misma. Estas forman parte ya de código de la plataforma y son descargadas, configuradas e instaladas automáticamente por Maven. En caso de necesitar añadir alguna librería nueva, decisión que se tomará a lo largo de esta fase de diseño, será incluida en el fichero de configuración de Maven para que realice estos proceso de forma automática.

Hibernate

Como ya se ha indicado varias veces, k-PAX utiliza Hibernate que es un ORM (Object / relational mapping), es decir, una librería que permite representar en clases la estructura de una BD del tipo entidad relación. Es un servicio poderoso y de alto rendimiento para la persistencia de objetos relacionales y consultas de bases de datos. Su característica más especial es que evita escribir las consultas SQL en código lo que hace a la aplicación que la usa ser más portátil, aunque también permita escribirlas explícitamente.

A grandes rasgos Hibernate es una librería que guarda en archivos XML información sobre la estructura de la base de datos y se encarga de vincular una clase Java con una tabla de la base de datos y de implementar las típicas operaciones de inserción, eliminación, modificado o selección para la misma tabla, independientemente de las demás tablas con las que se relaciones y encargándose de los problemas de concurrencia.

Hibernate implementa JPA (Java Persistence API), que es la API de Persistencia para el lenguaje Java. JPA Se trata de una API que marca las pautas para poder representar una BD en clases Java.

En concreto habrá que mantener la variante de Hibernate denominada Hibernate Annotations, que se encuentra ya integrada en el código actual de k-PAX. Hibernate Annotations utiliza las anotaciones definidas en la JPA y elimina la necesidad de utilizar ficheros XML, como en las versiones iniciales de Hibernate, para guardar la información de la estructura de las tablas de la base de datos que contienen en las clases de la persistencia, mediante una serie de anotaciones en su código.

Hay que tener en cuenta la utilización de Hibernate Annotations para el diseño de la capa de persistencia ya sea para el diseño de las clases Java como para el de los archivos de configuración locales. En cualquier caso, la utilización de las anotaciones de Hibernate Annotations quedará restringido a los casos en que no se encuentre otra manera de representar la estructura de la base de datos con las anotaciones definidas para la JPA en el paquete javax.persistence. De modo que se evite en la medida de lo posible ligar el código con un ORM concreto (Hibernate).

Descomposición en clases

Entidades del sistema

En el apartado de análisis de clases en la fase de análisis se daba una pequeña aproximación de dichas entidades básicas que necesitaría el sistema para trabajar correctamente. Posteriormente, al hablar de la capa de persistencia, cada entidad se corresponderá con una o varias clases en cada paquete, de los declarados en el apartado de descomposición física y que posiblemente sean ya existentes.

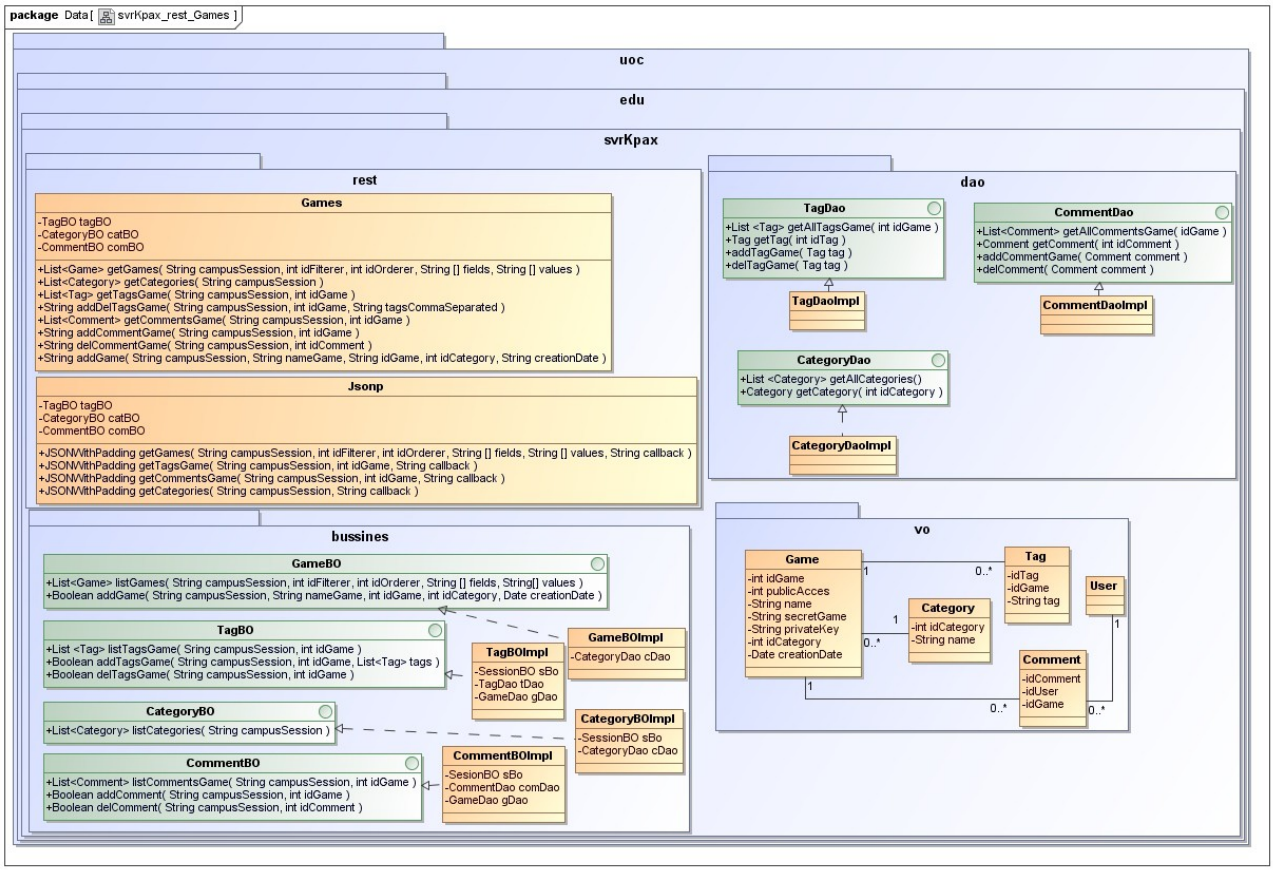
Diagrama de clases

Durante el análisis se han descrito los requisitos que debe cumplir el sistema. A partir de ellos se han podido establecer los casos de uso que deban cumplir dichos requisitos. Para que la aplicación pueda cumplir dichos casos de uso, se ha de diseñar una interfaz o interfaces que definan las funciones o servicios web que serán accedidos con elgg para la capa de presentación y cuyas implementaciones se sirvan de las clases e interfaces de la capa de persistencia. Estas interfaces y sus clases seguirán la estructura de paquetes definida en el apartado anterior de descomposición física.

A continuación se muestra el diagrama de clases diseñado. Se han producido diversos problemas durante el transcurso del curso académico que han retrasado inevitablemente la entrega del proyecto, por eso, el proyectante calcula que no tiene tiempo suficiente para desarrollar con tiempo la gestión de metadatos, y ha decidido eliminarla. En consecuencia, las clases y funciones que darían soporte a la gestión de metadatos no se han definido.

Si posteriormente al desarrollo de las funcionalidades principales del proyecto se dispone de tiempo suficiente, se recuperará la idea principal de desarrollarla planteándolo como un ciclo incremental.

El diagrama siguiente contiene las nuevas clases e interfaces diseñadas para dar soporte a las nuevas funcionalidades. Las clases e interfaces existentes no mostradas en este diagrama se mantendrán como hasta ahora. Mientras que las clases e interfaces que existen ya y que sí se muestran, definen nuevos campos o funciones. Además para algunas de esas clases existentes también se modifican algunas funciones añadiéndoles más parámetros, necesarios para el desarrollo del proyecto.



Las modificaciones son las siguientes:

- Capa de persistencia:

  A. En el paquete *uoc.edu.svrKpax.dao* se han definido las siguientes interfaces:

  i. *TagDao*, con las funciones siguientes:

  a) *getAllTagsGame*: devuelve el listado de tags del juego con id *idGame*.

  b) *getTag*: devuelve la información del tag con id *idTag*.

  c) *addTagGame*: Almacena la información de un tag.

  d) *delTagsGame*: Elimina todos los tag de un juego con id *idGame*.

  ii. *CategoryDao*, con las funciones siguientes:

  a) *getAllCategories*: devuelve el listado de todas las categorías de juegos posibles.

  b) *getCategory*: Devuelve la información de la categoría con id *idCategory.*

  iii. *CommentDao*, con las funciones siguientes:

  a) *getAllCommentsGame*: Devuelve el listado de todos los comentarios de un juego con id *idGame*.

  b) *getComment*: Devuelve la información de un comentario con id *idComment*.

  c) *addComment*: Almacena la información de un comentario.

  d) delComment: Elimina un comentario.

  iv. Sus respectivas implementaciones: *TagDaoImpl*, *CommentDaoImpl* y *CategoryDaoImpl*.

  A. En el paquete *uoc.edu.srvKpax.vo*: En este paquete se han definido las clases que

permitirán realizar el intercambio de la información de cada entidad entre el resto de interfaces y clases.

i. Clase *Game*: Se ha añadido dos nuevos campos: *idCategory* para poder determinar la categoría de un juego y *creationDate*, que indica la fecha en que se añade el juego a Kpax, para poder realizar ordenaciones por los juegos más nuevos.

ii. Clase *Tag*: Que incluye un campo *idTag* para establecer su identificador, el campo *idGame* para indicar el identificador del juego al que se asocia el tag y el valor del tag que se establece en el campo *tag*.

iii. Clase *Category*: Que incluye un campo *idCategory* para establecer su identificador y el valor de la categoría en el campo *name*.

iv. Clase *Comment*: Que incluye un campo *idComment* para establecer el identificador de un comentario, el campo *idUser* para relacionar el comentario con el identificador de un usuario y el campo *idGame*, para relacionar el comentario con el identificador d un juego. No se ha incluido el texto del comentario ni fecha ni demás información del comentario porque no se ha considerado necesario para cuantificar el número de comentarios por juego.

v. Clase *User*: No se ha modificado. Sólo se muestra en el diagrama para representar la relación entre la entidad de comentario y la de usuario.

- Capa de lógica:

A. En el paquete uoc.edu.srvKpax.bussines.

i. Interfaz GameBO, con las funciones:

a) listGames: Devuelve el listado de los juegos que cumplen el filtro con identificador idFilterer, ordenando el resultado por el método de ordenación con identificador idOrderer, a un usuario cuya sesión campusSession sea correcta.

Para facilitar la implementación de los distintos tipos de filtros y de ordenaciones, así como su extensión, se han añadido dos parámetros array que permiten introducir campos con valores adicionales: *fields* y *values*, respectivamente.

Así, por ejemplo, si se quiere filtrar el listado por el nombre de un juego estos campos pueden contener un array de String de tamaño 1 con la cadena “name” y otro array del mismo tamaño con el valor del nombre del juego. Con este diseño el número de campos-valor adicionales podrá ser infinito.

Los posibles valores de filtros son:

- 0: Sin filtrar. Mostrar todos.
- 1: Por categoría. Se deberá mandar un campo-valor de nombre *category* con el número identificador de la categoría.
- 2: Por nombre. Se deberá mandar un campo-valor de nombre *name* con el nombre del juego a filtrar.
- 3: Similares a otro juego: Se deberá mandar un campo-valor de nombre *idGame* con el identificador del juego que sirve de base para buscar juegos similares.

Los posibles valores de ordenaciones son:

- 0: Sin aplicar ningún tipo de ordenación.

- 1: Los más jugados.
- 2: Los más populares.
- 3: Los más nuevos.
- 4: Los más comentados.
- 5: Los más activos actualmente, tendrá en cuenta el número de instancias de juegos activos del momento.

b) *addGame*: Se ha modificado la función *addGame*, para que acepte como parámetros adicionales obligatorios el identificador de la categoría por medio del parámetro *idCategory* y la fecha en la que se añade el juego a kPAX, *creationDate*. El formato de entrada será *dd-MM-yyyy hh:mm:ss*.

ii. Interfaz *TagBO*:

a) *listTagsGame*: devuelve el listado de tags de un juego con identificador *idGame* si la sesión *campusSession* es válida.

b) *addTagsGame*: Añade un listado de *tags* al juego con identificador *idGame*, si la sesión *campusSession* es válida.

c) *delTagsGame*: Elimina todos los tags de un juego con identificador *idGame*, si la sesión *campusSession* es válida.

iii. Interfaz *CategoryBO*:

a) *listCategories*: Devuelve el listado de Categorías disponibles en el sistema.

iv. Interfaz *CommentBO*:

a) *listCommentsGame*: devuelve el listado de comentarios de un juego con identificador *idGame* si la sesión *campusSession* es válida.

b) *addComment*: Añade un comentario al juego con identificador *idGame*, si la sesión *campusSession* es válida.

c) *delComment*: Elimina el comentario con identificador *idGame*, si la sesión *campusSession* es válida.

v. Sus respectivas implementaciones:

a) GameBOImpl

- Se ha añadido una variable *CategoryDao* para poder comprobar si la categoría indicada existe, desde esta función.

b) Tag*BOImpl*

- Se han añadido variables *SessionBO*, siguiendo el ejemplo de las clases existentes en este paquete, para poder comprobar si la sesión es válida; *TagDao*, para poder realizar las operaciones de consulta o modificación en la capa de persistencia; y *GameDao*, para poder comprobar si el identificador del juego, *idGame* pertenece o no a un juego existente.

c) CategoryBOImpl

- Se han añadido variables *SessionBO*, siguiendo el ejemplo de las clases existentes en este paquete, para poder comprobar si la sesión es válida; y *CategoryDao*, para poder realizar las operaciones de consulta o modificación en la capa de persistencia.

d) *CommentBOImp*l

- Se han añadido variables *SessionBO*, siguiendo el ejemplo de las clases existentes en este paquete, para poder comprobar si la sesión es válida; *CommentDao*, para poder realizar las operaciones de consulta o modificación en la capa de persistencia; y *GameDao*, para poder comprobar si el identificador del juego, *idGame* pertenece o no a un juego existente.

B. En el paquete uoc.edu.srvKpax.rest

i. Clase *Games*:

a) *getGames*: Devuelve el listado de los juegos que cumplen el filtro con identificador *idFilterer*, ordenando el resultado por el método de ordenación con identificador *idOrderer*, a un usuario cuya sesión *campusSession* sea correcta.

Para facilitar la implementación de los distintos tipos de filtros y de ordenaciones, así como su extensión, se han añadido dos parámetros array que permiten introducir campos con valores adicionales: *fields* y *values*, respectivamente.

Así, por ejemplo, si se quiere filtrar el listado por el nombre de un juego estos campos pueden contener un array de String de tamaño 1 con la cadena "name" y otro array del mismo tamaño con el valor del nombre del juego. Con este diseño el número de campos-valor adicionales podrá ser infinito.

Los posibles valores de filtros son:

- 0: Sin filtrar. Mostrar todos.
- 1: Por categoría. Se deberá mandar un campo-valor de nombre id*Category* con el número identificador de la categoría.
- 2: Por nombre. Se deberá mandar un campo-valor de nombre *name* con el nombre del juego a filtrar.
- 3: Similares a otro juego: Se deberá mandar un campo-valor de nombre *idGame* con el identificador del juego que sirve de base para buscar juegos similares.

Los posibles valores de ordenaciones son:

- 0: Sin aplicar ningún tipo de ordenación.
- 1: Los más jugados.
- 2: Los más populares.
- 3: Los más nuevos.
- 4: Los más comentados.
- 5: Los más activos actualmente, tendrá en cuenta el número de instancias de juegos activos del momento.

b) *addGame*: Se ha modificado la función *addGame*, para que acepte como parámetros adicionales obligatorios el identificador de la categoría por medio del parámetro *idCategory* y la fecha en la que se añade el juego a kPAX, *creationDate*. El formato de entrada será *dd-MM-yyyy hh:mm:ss*.

c) *getCategories*: Devuelve el listado de categorías disponibles en el sistema si la sesión *campusSession* es válida.

d) *getTagsGame*: Devuelve los tags de un juego con identificador *idGame*, si la sesión *campusSession* es válida.

e) *addDelTagsGame*: elimina los tags de un juego con identificador *idGame* y posteriormente añade los tags del parámetro *tagsCommaSeparated* al juego siempre que se indique una sesión *campusSession* válida.

Los tags deben estar contenidos en una única cadena, separados por comas. El sistema se encargará de separarlos para añadirlos posteriormente, de forma individual, al sistema.

f) *getCommentsGame*: Devuelve el listado de comentarios de un juego con identificador *idGame*, siempre que la sesión *campusSession* sea válida.

g) *addCommentGame*: Añade un comentario a un juego con identificador *idGame*, siempre que la sesión *campusSession* sea válida.

h) *delCommentGame*: Elimina el comentario con identificador *idComment*, siempre que la sesión *campusSession* sea válida.

i) Se han añadido variables CommentBO, TagBO y CategoryBO, para poder realizar llamadas a las funciones de la capa de negocio desde la implementación de las funciones anteriores.

ii. Clase *Jsonp*: Para que elgg pueda realizar las llamadas por medio de ajax, se han modificado y añadido las funciones *getGames*, *getTagsGame* y *getCommentsGame*, descritas para la clase *Games*, anterior. Todas las funciones requieren del parámetro *callback* para funcionar bajo Json.

a) Se han añadido variables CommentBO, TagBO y CategoryBO, para poder realizar llamadas a las funciones de la capa de negocio desde la implementación de las funciones anteriores.

Capa de persistencia

Ficheros de configuración

A hablar de la capa de lógica ya se han identificado las entidades que se deberán almacenar en la BD. En el apartado de descomposición física, se indicaba que la capa de persistencia contaría con una serie de ficheros de configuración

Los ficheros de configuración serán dos:

1. Hibernate.cfg.xml: Que contiene los datos de acceso a la base de datos para facilitar la portabilidad de las aplicaciones independientemente de la base de datos a la que acceda. El uso de anotaciones de la JPA y en su caso de Hibernate Annotations reduce drásticamente el tamaño de este fichero que ya no necesita contener la estructura de la base de datos para relacionarla con las clases de intercambio, sino que esta estructura se encuentra definida en las mismas clases de intercambio por medio de las anotaciones.
2. ApplicationContent: Las variables GameBO, CommentBO, TagBO y CategoryBO de la clase Games y Jsonp y las variables SessionBO y GameDao, CommentDao, TagDao y CategoryDao de las clases del paquete uoc.edu.srvKpax.bussines deben ser inicializadas con objetos de sus clases implementadoras. El fichero ApplicationContent permite inicializar las variables de forma automática mediante sus funciones get y set.

   Por tanto será necesario indicar el nombre de las variables y sus clases implementadoras

para que se inicialicen de forma automática.

Diseño de base de datos

Como se ha indicado anteriormente, cada entidad actual tiene su representación en BD por medio de una tabla. Es decir, que cada entidad se corresponde con una única tabla. Existiendo además tablas que representan las relaciones entre entidades. En este sentido, hay que determinar las tablas de BD que se deban crear.

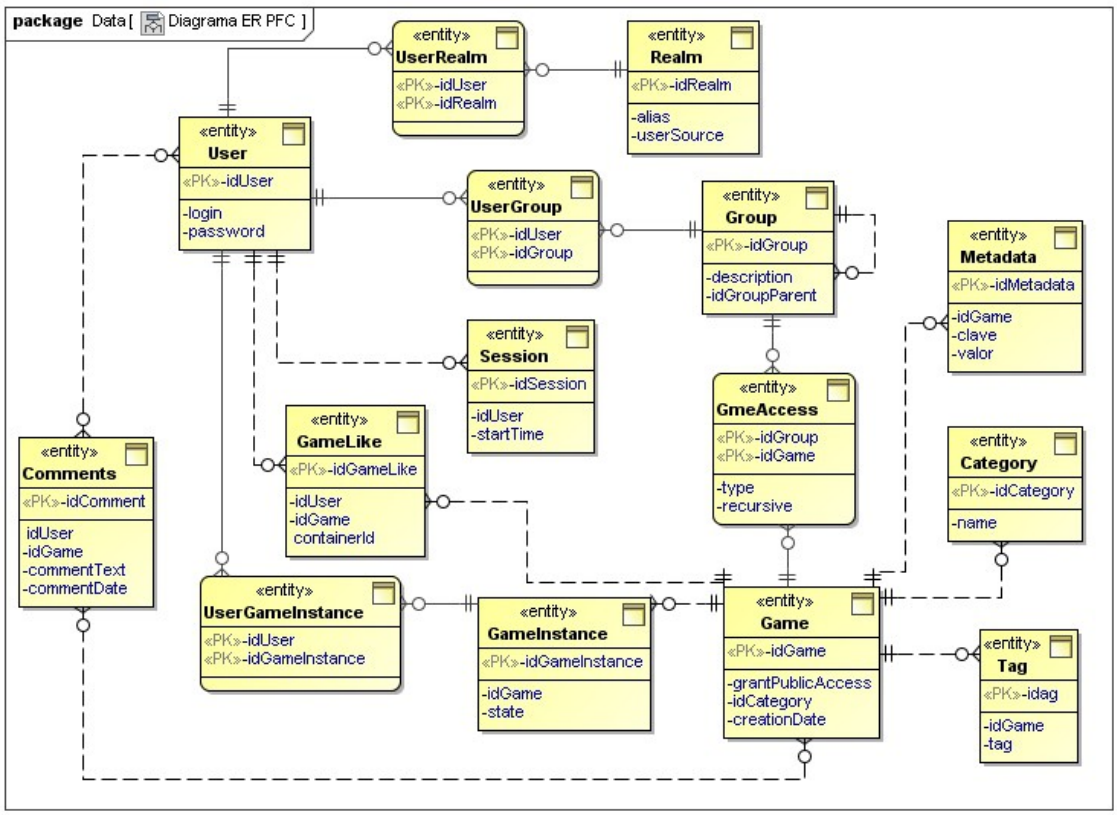
Para ello, primero se diseñará el esquema relacional de la BD por medio de un diagrama de entidad-relación, asignando una tabla a cada entidad, para , después transformar dicho esquema en tablas bien definidas. Posteriormente se aplicarán las técnicas de normalización de tablas que evitarán redundancia de datos y por último se optimizará la BD mediante la definición de índices en algunos de los campos de las tablas más usadas.

Como es requisito del sistema que sea compatible con el sistema anterior, deberán establecerse las diferencias y semejanzas entre la BD que se diseñe en este apartado con la actual de KPAX para poder definir un plan de transformación de datos que impida que se produzcan errores posteriores por falta de datos o similar. Es decir, si por ejemplo se determina que hay que añadir un nuevo campo a una tabla y este no puede ser nulo, en este plan se indicará el valor que deba tomar cada tupla de la tabla para que se cumpla la restricción.

Así mismo es requisito imprescindible, de toda BD, que se represente correctamente la realidad que se modela. Por lo que se tendrán que establecer los mecanismos de control necesarios para evitar posibles problemas de concurrencia.

Esquema relacional

El diagrama de entidades se ha mostrador en la fase de análisis. La figura siguiente muestra dicho diagrama:



El modelo relacional intenta acercarse al diseño real de la base de datos. Se puede obtener fácilmente a partir de las entidades definidas en la fase de análisis y que se corresponden con clases en el paquete uoc.edu.svrKPAX.vo, tal y como se muestra en el diagrama de clases UML.

Parte de este trabajo se hizo en la fase de análisis al determinar las entidades necesarias. No obstante, será tras terminar este apartado cuando las tablas y campos de la base de datos, claves principales, campos únicos, claves externas y foráneas; sean definitivas. Este modelo relacional servirá para diseñarlos.

Hasta ahora se han identificado las entidades del sistema y sus relaciones, pero, estas últimas se han descrito vagamente. Para poder transformar el esquema relacional de la BD del sistema en un conjunto de tablas con sus campos, índices, claves foráneas, etc. será necesario definirías primero:

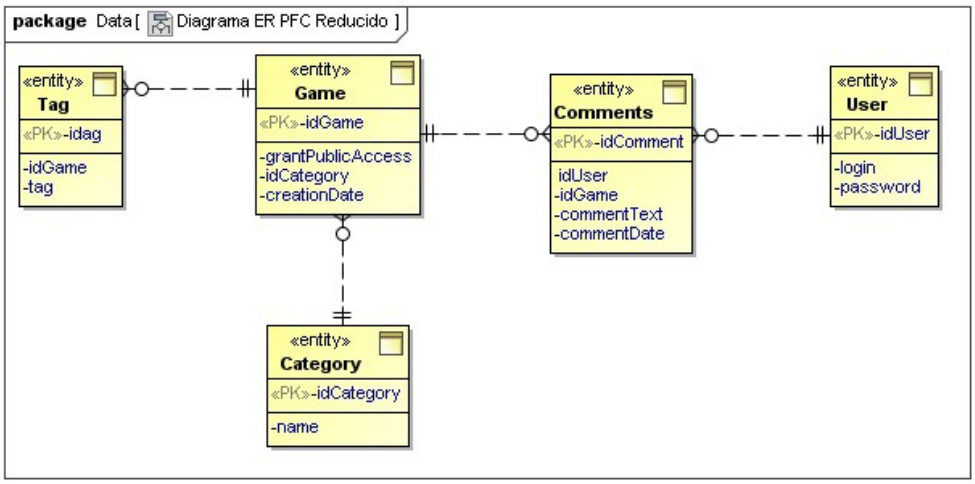
1. Un juego puede tener estar definido por cero o varios tags y un mismo tag puede describir a 1 o más juegos.
2. Un juego puede tener, o no, varios comentarios de uno o más usuarios, pero un mismo comentario sólo puede pertenecer a un juego. Del mismo modo un mismo comentario sólo pertenece a un usuario, mientras que un usuario puede comentar, o no, uno o varios juegos
3. Una categoría puede describir, cero o más juegos, mientras que un juego siempre se definirá con una única categoría. Si se desea describir el juego como hibrido de otras categorías se podrá hacer con tags, mientras que el valor de la categoría sería la más representativa del juego.

Ahora ya se sabe cuales son la relaciones entre las entidades de la BD, pero, antes de poder definir las tablas de la base de datos a partir del esquema relacional de la BD del sistema aún queda por conocer las claves primarias y secundarias de la BD.

Para evitar tener que declarar claves primarias débiles, se seguirá la estrategia utilizada hasta ahora que es la de utilizar un único campo autoincremental que sirva de clave primaria. Así:

1. La tabla Game seguirá teniendo como clave primaria el identificador idGame.
2. La tabla User seguirá teniendo como clave primaria el identificador idUser.
3. La tabla Category tendrá una clave denominada idCategory.
    a. El campo name de cada categoría es único entre todos los demás, de modo que forma una clave secundaria.
4. La tabla Tag tendrá una clave denominada idTag.
    a. La combinación de los campos idGame y tag son únicos, de modo que forman una clave secundaria.
5. La tabla Comment tendrá una clave denominada idComment.

La figura siguiente muestra el diagrama de tablas de la BD. Como se puede ver no ha diferido de la prevista en la fase de análisis. Sólo se muestran las tablas de las que se han hablado:



Descripción tablas

Una vez que se ha definido el esquema relacional en el apartado anterior, se puede diseñar el modelo de tablas de la BD. Para hacerlo se utilizarán las reglas de transformación de un esquema relacional a un modelo de tablas que se impartieron en la asignatura de Diseño de Base de Datos durante el primer ciclo de los estudios universitarios del proyectante.

Esas reglas explican que se crea una tabla por cada entidad con clave fuerte. Dado que todas las clases del diagrama de clases UML de la BD tienen claves fuertes, cada una de las clases se transformará en una tabla de la BD.

Los nombres de las tablas serán los mismos que la de las clases de la capa de persistencia que modelen la misma entidad, estos son los que se muestran en el diagrama. Los de cada columna serán los mismos que la de los atributos de dichas clases.

Las demás reglas de transformación dicen lo siguiente:

- Si hay una relación entre dos entidades y el vínculo es 1:1, se creará una columna en una de las entidades con la clave, haciendo relación externa hacia la otra entidad.
- Si una relación es de granularidad 1:N, también se creará una columna en la entidad de granularidad 1, con la clave, y una relación externa hacia la entidad de granularidad N.
- Si la relación tiene un tipo de vinculo M:N, entonces se creará una nueva tabla que en nuestro caso se denominará con la mezcla de los nombres de las dos entidades que relaciona, unidos por un guion bajo. Dicha tabla contendrá columnas con las claves de las dos entidades y harán relaciones externas hacia sus respectivas entidades.

Todas las claves secundarias se transformarán en columnas con índices únicos.

Siguiendo por tanto las reglas de transformación del esquema relacional de la BD del sistema al modelo de tablas, se definen las siguientes tablas:

1. Tabla **Game**:
    1. *idGame*: Clave de la tabla
    2. *name*
    3. *grantPublicAccess*
    4. *secretGame*
    5. *privateKey*

6. *idCategory:* CE hacia *Category.*
7. *CreationDate*

2. Tabla **Category**:
    1. *idCategory*: Clave de la tabla
    2. *name:* Clave secundaria de la tabla. Índice Único.
3. Tabla **Tag**:
    1. *idTag*: Clave de la tabla.
    2. *idGame*: CE hacia *Game*.
    3. *tag*
4. Tabla **Comment**:
    1. *idComment*: Clave de la tabla.
    2. *idUser: CE hacia User.*
    3. *idGame: CE hacia Game.*
    4. *commentText*
    5. *commentDate*

Normalización

El siguiente paso para el diseño de la BD del sistema es la normalización, que prepara la BD de tal manera que se elimina la redundancia de datos en la BD. El objetivo es dejar la BD en la cuarta forma normal o Forma Normal de Boyce Codd (FNBC) que es el mayor nivel de normalización. Aunque en ocasiones es conveniente dejar la BD en niveles anteriores de normalización por cuestión de rendimiento. En caso de que esto ocurra se explicará porqué. En cualquier caso nunca el proyectante dejará la BD en un nivel anterior a la 3ª Forma Normal por tener demasiada redundancia de datos.

En cualquier caso, para la normalización se entenderá por BD las tablas modificadas/añadidas o afectadas por los cambios. Se entiende que el resto de tablas siguen la forma normal que tengan por cuestiones de diseños no contemplados en este proyecto.

Para empezar a normalizar es necesario detectar todas las dependencias funcionales (DFs) de todas las tablas de la BD. Una dependencia funcional es toda aquella columna de una tabla que en relación a otra columna al conocer el valor de la primera sólo puede haber un único valor de la segunda. Obtener las DFs de la BD es trivial por lo que no se expondrán en este documento.

Primera Forma Normal (1FN)

Una relación R está en 1FN si todos sus atributos son mono-valuados.

Por tanto para saber si todas las relaciones de la BD están en 1FN será necesario identificar los atributos multivalor.

Como no existen tablas que contengan atributos multivalor en la BD por lo que la BD está en 1FN.

Segunda Forma Normal (2FN)

Una relación R está en 2FN si y sólo si está en 1FN y todo atributo no primo depende

funcionalmente de manera total de toda clave de R (primaria o candidata).

Los atributos no primos son todos aquellos que no son clave.

Para saber si todas las relaciones están en 2FN hace falta identificar los atributos no primos.

Como todos los atributos primos dependen funcionalmente de forma completa de alguna clave, y todas están en 1FN, también están en 2FN.

Tercera Forma Normal (3FN)

Una relación R está en 3FN si y sólo si está en 2FN y para toda DF de X relacionado con Y de R, X es superclave o bien Y es atributo primo.

Un atributo es superclave si forma parte de una clave

Por tanto es necesario identificar los atributos no primos que dependen funcionalmente de un atributo que no es superclave.

Como no existe ningún caso con esta situación y todas las relaciones están en 2FN, entonces también están en 3FN.

Forma Normal de Boyce-Codd (FNBC)

Una relación R está en FNBC si y sólo si está en 3FN y para toda DF de X relacionado con Y de R, X es superclave.

Para saber si todas las relaciones están en FNBC es necesario identificar los atributos no superclave para estudiarlos.

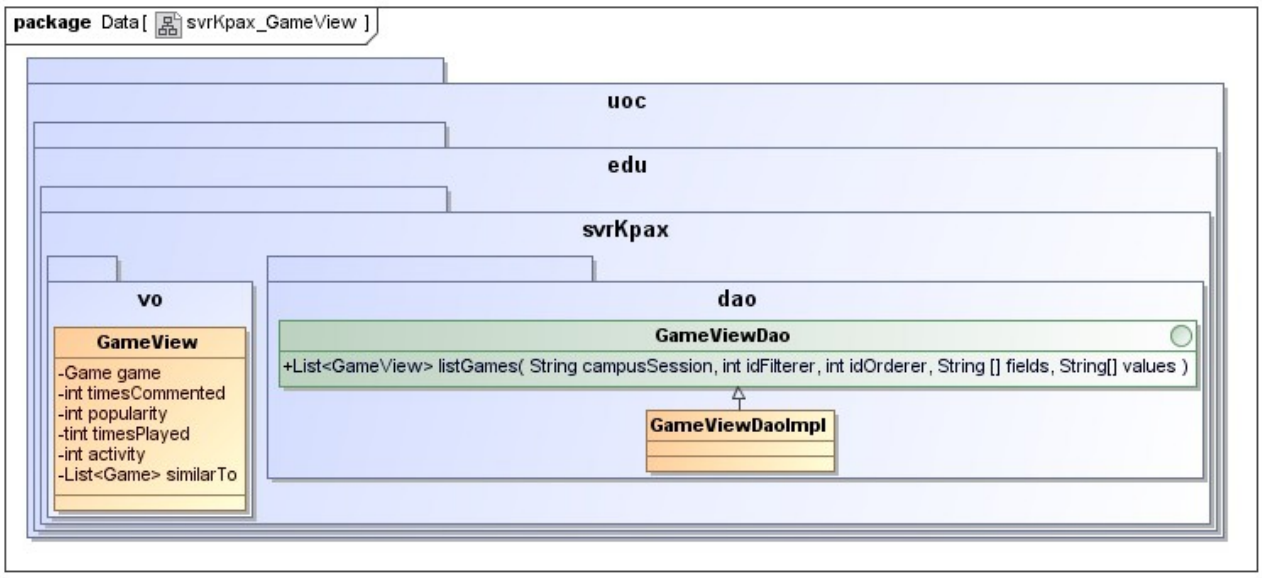
Como todos los atributos de todas las DFs dependen funcionalmente de una superclave, no existen DFs que no sean superclave, y todas las relaciones están en 3FN, entonces, también están en FNBC y por tanto se puede decir que la BD está normalizada en FNBC.

Vistas

Ante la necesidad de realizar consultas sobre diferentes tablas para poder ordenar por los más jugados, que requiere mirar en la tabla de GameInstance; los más populares, que requiere mirar en la tabla de GameLike; los más nuevos, que requieren mirar en la tabla Game; los más comentados, que requieren mirar en la tabla Comments; y los más activos, que una vez más requieren mirar en la tabla de GameInstance; además de los filtros de similares, que requieren mirar en la tabla de Tags y de Categoría; por categoría que una vez más requiere mirar en la tabla de Game; y por nombre que una vez más requiere mirar en la tabla de Game.

Añadir una vista con estos datos supondría varias mejoras. La primera que se tiene toda la información en una sola vista, lo que simplifica enormemente las consultas SQL. La segunda, que se optimizan las consultas porque el SGBD puede determinar gran parte del algoritmo de búsqueda antes de conocer la consulta SQL final.

Pero crear una vista requiere algo más aparte de la vista en sí misma. Esto es crear clases para que se pueda acceder a su información. En la siguiente imagen podrá ver el diagrama de clases diseñado para dar soporte a esta vista que se denominará GameView.



Se han diseñado las siguientes clases e interfaz:
En el paquete uoc.edu.svrKpax.vo, la clase GameView que contiene una referencia a la clase Game, que contiene toda la información de un juego concreto, y el resto de atributos timesCommented, con el número de comentarios del juego game, popularity con el número de votos GameLike del juego, activity, con el número de jugadores jugando en el instante de la carga de datos y la lista de juegos similares, ordenados de mayor a menor similitud (el número de tags, y categoría, coincidentes).

En el paquete uoc.edu.svrKpax.dao, la interfaz GameViewDao y su clase implementadora GameViewDaoImpl. Como todas las interfaces de este paquete contiene las funciones de la capa de persistencia públicas para la capa de lógica.

Esta interfaz contiene una sola función. Al encapsular una vista, no es lógico la presencia de funciones que alteren los datos de la BD, tales como de adición, eliminación o modificación, por eso sólo es posible la existencia de funciones de consulta. En este sentido se ha diseñado una función que requiere los mismos parámetros que los de getGames de la clase Games del paquete uoc.edu.svrKpax.rest. La diferencia es que devuelve un listado de GameView en vez de Game. Esta función será utilizada desde la clase GameBO.

Análisis de Índices

Sabiendo que el 20% del total de transacciones que se pueden hacer a una BD suponen el 80% de las transacciones totales que se hacen en un periodo de tiempo determinado (Principio de Pareto o regla 80-20) es aconsejable optimizar ese 20% de transacciones para que el sistema sea eficiente.

Optimizar las BD es posible si se diseñan índices secundarios adecuados en las tablas a las que se acceden en las transacciones más frecuentes. Para ello, primero se han de analizar las transacciones más frecuentes para saber qué campos son los más consultados. De este modo, el proyectante ha identificado las transacciones más frecuentes del sistema a partir de la frecuencia de los casos de uso identificados durante el análisis.

Como se ha decidido no implementar la parte de metadatos, los más frecuentes son los relacionados con la búsqueda de juegos, ordenación, filtrado y consulta de la ficha de un juego. Por tanto, Las transacciones más frecuentes y que suponen aproximadamente el 20 % de transacciones del total de transacciones posibles del sistema son el filtro por categoría, filtro por nombre, ordenación por popularidad:

1. select * from GameView where Name = '…'
2. select * from GameView where idGame = '…'
3. select * from GameView orderBy Popularity

Las transacciones están ordenadas de más frecuentes a menos para aquellas transacciones que actúen sobre una misma tabla. Una vez que se han identificado las transacciones más frecuentes se pueden analizar cada una de ellas para saber los atributos y tablas a las que accede cada transacción y poder hacer índices secundarios en consecuencia:

Tabla	Transacción	Atributos
GameView	1	Where: name
	2	Where: idGame
	3	Order by: popularity

Una vez identificados los atributos que influyen en el rendimiento de las transacciones, hay que decidir los índices secundarios de las tablas.

Para todos los atributos clave se definen los índices primarios.

Para la tabla GameLike se define un índice de agrupación sobre los atributos popularity, idGame y name en el orden indicados.

De este modo, la organización de los índices en las tablas queda de la siguiente manera:

Tabla	Trans.	Atributos	Org. Primaria Índices secundarios
GameView	1-3	Where: name → a Where: idGame → a Group By: popularity → a	a. IP-c(id) ->clave secundaria

Eliminación en cascada

Muchas de las tablas diseñadas para la BD tienen algún campo con relación externa hacia otra tabla. Si una primera tupla es apuntada (relacionada) desde una segunda tupla de una tabla diferente cuyo campo tiene relación externa hacia la primera, al eliminarse la primera, la segunda tupla apuntaría hacia una tupla inexistente, lo que dejaría la BD en un estado inconsistente.

Actualmente no existe ningún mecanismo para evitar la no eliminación de tuplas dependientes. Hibernate puede realizar la supresiones de forma automática, sin embargo no existen anotaciones @OneToOne, @OneToMany, @ManyToOne ni @ManyToMany junto con las propiades cascade que permitiría implentarlo, por tanto. De hacerse automáticamente habría que modificarse probablemente todas las clases del paquete vo o por lo menos revisarse.

Esto supone un esfuerzo que va más allá de los límites del proyecto puesto que supondría probar toda la aplicación. En consecuencia la eliminación de tuplas dependientes se realizará a mano en cada una de las funciones de eliminación.

Casualmente sólo hay una entidad de la que dependan otras, esta es la de Game. Por tanto, al usar la

función delGame, habrá que eliminar también todas las entidades dependientes, estas son, tags y comentarios. También habría que eliminar GameLikes, Game instances, Records, etc. pero esto nuevamente se sale de los límites del proyecto porque suponen probar partes que se supone no se deben modificar.

Optimización

En este apartado se explican las decisiones tomadas para optimizar el sistema. Con objeto de optimizar las consultas a la BD ya se ha hablado del análisis de índices en el apartado anterior, de Diseño de BD, por lo que no se volverá a hablar de ello.

Pool de conexiones

El tiempo de establecer una conexión con la base de datos, es quizás una de los procesos más costosos de la ejecución de consultas a una base de datos. De modo que en un entorno en el que se realizan muchas transacciones a una base de datos puede verse ralentizada por tener que abrir conexiones repetidamente.

Hibernate permite ser configurado para utilizar un pool de conexiones en el fichero Hibernate.cfg.xml. Añadir un pool de conexiones permite mantener abiertas varias conexiones a la base de datos, aunque estas no se utilicen y quedar a la espera de que se solicite una nueva conexión para realizar una transacción. En ese momento Hibernate recupera una conexión abierta del pool, en vez de abrir una nueva, reduciendo considerablemente su tiempo de ejecución.

Configurar un pool de conexiones puede ser beneficioso para la plataforma en su conjunto. Por otro lado es requisito imprescindible mantener el diseño de la estructura así que no se implantará. No obstante se propone como mejora para la plataforma.

Seguridad

Un requisito muy importante del sistema, que se desarrolle con el proyecto, es que sea seguro.

En este apartado se diseñarán los mecanismos principales de seguridad del sistema y también se describirá más a fondo el mecanismo de validación, ya que aún no se ha descrito en profundidad.

Encriptación de las comunicaciones entre elgg y JBoss

En el apartado de descomposición física se indica que la comunicación entre elgg y la parte servidora se realiza por medio de diferentes servidores web: xampp o lampp y JBoss. Actualmente estos servidores se encuentran en la misma máquina pero no podemos fiarnos de que sea siempre así, puesto que pueden ser separados en máquinas diferentes, por eso, la comunicación entre ambos debe ser segura.

Por otro lado, de esto se encargan el módulo de seguridad, descrito en la documentación en inglés de la arquitectura de Kpax. Este módulo realiza la encriptación de datos con el algoritmo de encriptación AES. Elgg se encarga de encriptar los datos con la clave pública obtenida con el módulo apiAdmin de elgg mientras que en la parte Java, se desencripta con la función decrypt de la clase uoc.edu.svrKpax.util.AES.java y la clave privada definida en uoc.edu.svrKpax.util.ConstatsKPAX.java.

Validación de usuario

Para evitar que un usuario sin validar realice cambios en la base de datos, todas las funciones diseñadas deben comprobar que la sesión sea válida, por medio de la función validateSession de la clase uoc.edu.svrKpax.bussines.SessionBO.java antes de realizar ninguna llamada a las clases de la capa de persistencia. Por supuesto, en caso de que la sesión no sea válida toda respuesta tendrá que ser en forma de error indicando que la sesión no es válida.

Control de concurrencia

"Una base de datos de la que no te puedas fiar no es una base de datos."[1]
Si se hecha un vistazo al apartado de descomposición física, se puede recordar que el sistema debe poder descomponerse de tal manera que permita tener la BD de forma centralizada en un servidor de BD. Este requisito conlleva el problema de la aparición de problemas de concurrencia.

Cuando varios usuarios acceden simultáneamente a una base de datos para realizar cambios en ella es cuando se pueden dar los posibles problemas de concurrencia que pueden ocurrir durante la ejecución de sus operaciones. Los problemas que nos podemos encontrar son los de lecturas sucias, resúmenes incorrectos, actualizaciones perdidas, lecturas no repetibles, lecturas fantasmas y otros problemas.

Ante estos problemas se pueden tomar dos estrategias: el control de la concurrencia optimista, o pesimista. El primero, el optimista, consiste en pensar que no se van a dar estos problemas, y si se dan deshacer todos los cambios. La forma de determinar si se dan estos problemas es que se mantenga una versión de cada tupla de la base de datos e ir aumentando su número con cada modificación. Si un usuario lee los datos de una tupla e intenta cambiar sus datos, si el número no coincide con el leído es que otro usuario más rápido ha modificado los datos entre medio, de modo que el primer usuario tendrá que releer los datos y volver a intentarlo o desistir.

El segundo consiste en pensar lo contrario, que siempre se darán, de modo que se de permiso para consultar y modificar la base de datos a un primer usuario, bloqueando a los demás usuarios aquello que el primero quiera cambiar para desbloquearlo al terminar y dar permiso al siguiente usuario.

La estrategia optimista es más propia de entornos con pocas transacciones y pocos usuarios, porque es más rápida y la pesimista cuando hay muchas transacciones, por ser mas segura. Esta parece ser la estrategia diseñada para la plataforma de KPAX de modo que se mantendrá.

No parece que la arquitectura actual siga ninguna estrategia y la estrategia optimista requeriría introducir un campo valor en todas y cada una de las tablas, de modo que sólo se puede seguir la pesimista sin tener que realizar cambios en todas las tablas de BD y clases del paquete uoc.edu.svrKpax.vo.

Por otro lado, las cuestiones de diseño de la plataforma no atañen al proyecto de modo que lo mantendremos como está. No obstante, se recomienda al menos establecer un nivel de aislamiento READ COMMITED.

Nivel de aislamiento READ COMMITED

Para evitar que una operación que está modificando la BD y no ha terminado altere el resultado de otras operaciones, véase por ejemplo los problemas de lecturas sucias y de resúmenes incorrectos, se decidido que absolutamente todas las transacciones que se realicen a la BD tengan un nivel de

1 Cita expresada repetidas veces por el profesor Francisco José García Izquierdo en sus clases de Programación de Base de Datos (PRBD) de la Universidad de La Rioja mientras explica las bondades y los peligros de las prácticas de control de concurrencia.

aislamiento de READ COMMITED.

Hacer que todas las transacciones tengan un nivel de aislamiento read commited no implica una reducción significativa de rendimiento, ya que incluso algunos gestores de bases de datos (SGBD) como el de Oracle no admiten un nivel de aislamiento menos restrictivo. Utilizar ese nivel de aislamiento ayuda a resolver la gran mayoría de los problemas de concurrencia que se puedan encontrar.

Por ejemplo, todos los problemas de lectura sucia, de resumen incorrecto y los provocados porque se compruebe la existencia de una determinada tupla de una tabla y ésta halla sido modificada por otra transacción no confirmada de forma concurrente, no se ven afectadas.

Se propone como mejora de Kpax establecer un nivel de aislamiento READ COMMITED.

Restricciones de integridad

Todas las claves tanto primarias como secundarias de las tablas modificadas/creadas tienen índices sobre sus campos.

En muchas ocasiones los errores producidos por las restricciones de integridad de los índices o relaciones externas forman también un mecanismo de control de concurrencia que ayuda a solventar muchos de los problemas de concurrencia encontrados utilizando métodos de concurrencia optimista.

Los índices en las claves primarias unido con que éstas han de ser auto incrementales y si se establece el nivel de aislamiento read commited, forman un mecanismo más de control de concurrencia para los problemas de actualización perdida.

Un ejemplo:

Si se producen concurrentemente dos inserciones en una misma tabla, ambas han de calcular su identificador automáticamente y tomarían el mismo valor. Como el nivel mínimo de aislamiento es read commited, la segunda inserción se bloquearía hasta que la primera se confirmase o se abortase.

Al haber un índice de clave primaria en la tabla y el valor de éste ser igual para ambas inserciones, al desbloquearse la segunda transacción, se produciría un error de integridad sobre la clave primaría (si la primera transacción se confirmó) ya que la primera transacción insertó una tupla con el mismo identificador y éste no puede repetirse.

Como el error de restricción de integridad producido, en este caso, sería detectable, el paso a proceder sería el de reintentar la inserción recalculando el identificador de la tupla hasta que no se volviese a producir el mismo error.

Las relaciones externas de algunos campos de tablas hacia los campos clave de otras tablas, junto con el nivel de aislamiento read commited de todas las transacciones y de la eliminación en cascada de las tuplas con relación externa forman otro control de concurrencia.
Por todo ello una vez más se propone como mejora establecer el nivel de aislamiento read commited, como mínimo.

Capa de presentación

La capa de la presentación es la capa que se encarga de interactuar con el usuario para servir de puente entre él y el resto de la aplicación. Ofrece una interfaz gráfica de usuario amigable y sencilla en dónde representar la información que es procesada en la aplicación.

En el apartado de descomposición física, en este mismo apartado de diseño, se indica que la capa de presentación se descompone en ficheros PHP contenidos en módulos elgg que realizan comunicaciones con la capa de negocio por medio de los servicios web publicados en JBoss.

Descomposición en ficheros

Durante la fase de análisis se determinaron dos vistas a modificar: el listado de juegos y la ficha de un juego. En la primera vista, en el listado de juegos, se ejecutarían los casos de uso de Ordenar búsqueda y Filtrar búsqueda, implicando el listado de juegos.

En la segunda se realizarían la adición de tags, metadatos y la adición de los dos nuevos campos de un juego (categoría y fecha de creación). Por otro lado, ya se ha indicado que la gestión de metadatos se ha eliminado por falta de tiempo.

Ambas vistas ya están implementadas actualmente en elgg, de modo que habrá que determinar los ficheros a modificar y/o añadir teniendo en cuenta este dato.

La adición y consulta de la ficha de un juego se encuentra diseminada en los siguientes ficheros:

- Adición de un juego: El fichero elgg/mod/kpax/views/default/forms/kpax/save.php del servidor de elgg construye el formulario de entrada de datos. En este primer fichero se necesitarán añadir campos para ingresar los valores.

  Al pulsar sobre añadir se realiza una llamada a la acción elgg que introduce los datos en la base de datos de elgg y en la de kpax (por medio de una llamada al servicio web de adición de juegos). Este fichero se encuentra en la ruta /elgg/mod/kpax/actions/kpax/save.php. Desde este segundo fichero se deberá realizar una llamada a la función addGame y si el juego es añadido, añadir también sus tags con la función addDelTagsGame.

- Consulta de la ficha de un juego: El fichero elgg/mod/kpax/views/default/object/kpax.php muestra la ficha de un objeto de subtipo kpax, es decir un juego. Desde este fichero se deberá realizar una llamada a la función getGame y a la de getCategory, para obtener el nombre de la categoría.

La adición y eliminación de comentarios actualmente se realiza desde los ficheros siguientes:

- Adición: El fichero elgg/actions/comments/add.php añade un comentario en la base de datos de elgg. Se observa que el fichero no está en ningún módulo concreto sino que forma parte integra de elgg y que un comentario puede pertenecer a cualquier objeto. Será necesario comprobar previamente a la llamada a la función addCommentGame que el objeto sea de subtipo kpax.

- Eliminación: El fichero elgg/actions/comments/delete.php elimina un comentario concreto. Será necesario realizar una llamada a la función delCommentGame e igualmente comprobar antes de ello que el comentario pertenece a un objeto de subtipo kpax (un juego).

El listado de juegos se realiza desde los siguientes ficheros, en estos será necesario realizar una llamada a la función getGames e insertar los campos para ingresar los valores de los parámetros de la llamada, es decir, una lista desplegable con las posibles ordenaciones, otra con los posibles filtrados y un campo para ingresar el valor por el que filtrar. También se mostrará el listado de

categorías obteniéndolas por medio de la función getCategories.

La inserción de estos campos y del listado de categorías no ha sido posible añadirla finalmente por falta de tiempo. La situación final del proyecto se explicará en la fase de construcción.

- Listado de todos los juegos: El fichero elgg/mod/kpax/pages/kpax/all.php se corresponde con la pestaña de All del apartado de prototipado en la fase de análisis. Aquí se mostrará como máximo el total de juegos.
- Listado de juegos propios: El fichero elgg/mod/kpax/pages/kpax/owner.php se corresponde con la pestaña de Mine del apartado de prototipado en la fase de análisis. Aquí se mostrarán como máximo los juegos que hayan sido añadidos por el usuario.
- Listado de los juegos de amigos: El fichero elgg/mod/kpax/pages/kpax/owner.php se corresponde con la pestaña de Friends del apartado de prototipado en la fase de análisis. Aquí se mostrarán como máximo los juegos que hayan sido añadidos por los amigos del usuario.

Por último las llamadas a las funciones del servidor srvKpax se realizan desde el fichero elgg/mod/kpax/lib. Este fichero es el que se encarga de servir de enlace entre elgg y el servidor de kpax definiendo una librería de funciones, con una función PHP por cada una de los servicios web publicados en el servidor de kpax. En consecuencia, será necesario añadir una función por cada una de los servicios nuevos añadidos.

También será necesaria la creación de un fichero que construya el formulario para la adición de una categoría, que será almacenado en la ruta elgg/mod/kpax/views/default/forms/category/save.php, dos acciones que añadan los datos en elgg y en la base de datos de kpax /elgg/mod/kpax/actions/category/save.php y que los eliminen, /elgg/mod/kpax/actions/category/delete.php y una vista que muestre cada categoría en elgg/mod/kpax/views/default/object/category.php.

Como se explica en la fase de construcción, esta última parte no me ha dado tiempo a implementarla. De modo que en resumen, no se añaden fichero extra a los existentes. Tan sólo ha sido necesario modificar los ya presentes para permitir los cambios.

Diseño pruebas

En la fase análisis se hacía un listado del tipo de pruebas que se deben hacer a lo largo del desarrollo de proyecto. De entre ellas se habló de las pruebas unitarias de caja blanca y negra para detectar posibles problemas en la implementación de las funciones de las aplicaciones del proyecto.

Las pruebas de caja blanca no se podrán diseñar hasta la fase de implementación porque para diseñarlas es necesario conocer el código del programa. Sin embargo, para las de caja negra sólo se necesitan conocer las cabeceras de las funciones de las interfaces de la capa de lógica, información de la que ya se dispone.

Se hacen las pruebas a partir de las interfaces de la capa de lógica porque lo que se quiere probar es que los casos de uso del sistema se implementan correctamente, lo que implica que la lógica de negocio y la de persistencia, funcionen correctamente. No es imprescindible, por tanto, diseñar pruebas para los ficheros PHP, porque la forma como se representen los datos no es crítica.

En este apartado se diseñarán las pruebas unitarias de caja negra de la aplicación principal del proyecto. La siguiente tabla muestra una plantilla de una prueba unitaria de caja negra cualquiera que será rellenada por cada una de las pruebas que se describan:

PRUEBA UNITARIA X (válida /no válida)
Descripción:
Entradas:
Resultado esperado:

La primera fila contiene el nombre de la prueba. Indica además si la prueba es válida o no
La segunda fila la situación en la que se encuentre el sistema.
La tercera fila los valores de los parámetros de la función que se pruebe.
Y la cuarta fila el valor que se devuelve al ejecutarse la función.

No todas las pruebas que se realicen se describen en las líneas siguientes. A continuación, se indican las pruebas unitarias de caja negra agrupando por funciones de consulta, listado, adición, eliminación, modificación y otros grupos que se asemejen entre sí. Se diferenciarán entre las pruebas que den resultados válidos y los que no, es decir, entre aquellos que dan un resultado conforme a la finalidad para la que está diseñada la función y aquellos que no.

Pruebas de adición

Pruebas de unidad de addDelTagGame:

PRUEBA UNITARIA 1 (válida)
Descripción: Sesión válida, juego existente y contiene tags nuevos, pero no anteriores
Entradas: creationSession "xxxxx" idGame: 1 tagsCommaSeparated: “tag1,tag2”
Resultado esperado: “OK”

PRUEBA UNITARIA 2 (no válida)
Descripción: Sesión inválida, juego existente y contiene tags
Entradas: creationSession "yyyyy" idGame: 1 tagsCommaSeparated: “tag1,tag2”
Resultado esperado: “ERROR”

PRUEBA UNITARIA 3 (no válida)
Descripción: Sesión válida, juego inexistente y contiene tags
Entradas: creationSession "xxxxx" idGame: -1 tagsCommaSeparated: “tag1,tag2”
Resultado esperado: “ERROR”

PRUEBA UNITARIA 4 (no válida)
Descripción: Sesión válida, juego existente y no contiene tags
Entradas: creationSession "xxxxx" idGame: 1 tagsCommaSeparated: “”
Resultado esperado: “ERROR”

Las pruebas unitarias para addCommentsGame son similares, quitando el parámetro tagsCommaSeparated y la equivalente prueba unitaria 4 que no existiría. Las pruebas de addGame son:

PRUEBA UNITARIA 5 (válida)
Descripción: Sesión válida, nombre inexistente, juego inexistente, categoría existente y fecha de creación con formato “dd-MM-yyyy hh:mm:ss”
Entradas: creationSession "xxxxx" name: “aaaa”, idGame: 2, idCategory: 1, creationDate: “14-06-2012 19:45:27”
Resultado esperado: “OK”

PRUEBA UNITARIA 6 (no válida)
Descripción: Sesión inválida, nombre inexistente, juego inexistente, categoría existente y fecha de creación con formato “dd-MM-yyyy hh:mm:ss”
Entradas: creationSession "yyyyy" name: “aaaa”, idGame: 2, idCategory: 1, creationDate: “14-06-2012 19:45:27”
Resultado esperado: “ERROR”

PRUEBA UNITARIA 7 (no válida)
Descripción: Sesión válida, nombre inexistente, juego inexistente, categoría existente y fecha de creación con formato “dd-MM-yyyy hh:mm:ss”
Entradas: creationSession "xxxxx" name: “bbbb”, idGame: 2, idCategory: 1, creationDate: “14-06-2012 19:45:27”
Resultado esperado: “ERROR”

PRUEBA UNITARIA 8 (no válida)
Descripción: Sesión válida, nombre inexistente, juego inexistente, categoría existente y fecha de creación con formato “dd-MM-yyyy hh:mm:ss”
Entradas: creationSession "xxxxx" name: “aaaa”, idGame: 1, idCategory: 1, creationDate: “14-06-2012 19:45:27”
Resultado esperado: “ERROR”

PRUEBA UNITARIA 9 (no válida)
Descripción: Sesión válida, nombre inexistente, juego inexistente, categoría inexistente y fecha de creación con formato “dd-MM-yyyy hh:mm:ss”
Entradas: creationSession "xxxxx" name: “aaaa”, idGame: 2, idCategory: -1, creationDate: “14-06-2012 19:45:27”
Resultado esperado: “ERROR”

PRUEBA UNITARIA 10 (no válida)
Descripción: Sesión válida, nombre inexistente, juego inexistente, categoría existente y fecha de creación con formato incorrecto
Entradas: creationSession "xxxxx" name: “aaaa”, idGame: 2, idCategory: 1, creationDate: “14062012194527”
Resultado esperado: “ERROR”

PRUEBA UNITARIA 11 (no válida)
Descripción: Sesión válida, nombre inexistente, juego inexistente, categoría existente y fecha de creación con formato correcto “dd-MM-yyyy hh:mm:ss”, pero fecha incorrecta
Entradas: creationSession "xxxxx" name: “aaaa”, idGame: 2, idCategory: 1, creationDate: “50-13-2012 59:40:99”
Resultado esperado: “ERROR”

PRUEBA UNITARIA 12 (no válida)
Descripción: Sesión válida, nombre inexistente, juego inexistente, categoría existente y fecha de creación vacío
Entradas: creationSession "xxxxx" name: “aaaa”, idGame: 2, idCategory: 1, creationDate: “”
Resultado esperado: “ERROR”

PRUEBA UNITARIA 13 (no válida)
Descripción: Sesión válida, nombre vacío, juego inexistente, categoría existente y fecha de creación con formato “dd-MM-yyyy hh:mm:ss”
Entradas: creationSession "xxxxx" name: “”, idGame: 2, idCategory: 1, creationDate: “14-06-2012 19:45:27”
Resultado esperado: “ERROR”

Pruebas de eliminación

Prueba unitaria para addDelTagsGame:

PRUEBA UNITARIA 14 (válida)
Descripción: Sesión válida, juego existente y contiene tags nuevos, además de anteriores.
Entradas: creationSession "xxxxx" idGame: 1 tagsCommaSeparated: “tag1,tag2”
Resultado esperado: “OK”

Pruebas unitarias para delCommentGame:

PRUEBA UNITARIA 15 (válida)
Descripción: Sesión válida, comentario existente
Entradas: creationSession "xxxxx" idComment: 1
Resultado esperado: “OK”

PRUEBA UNITARIA 16 (no válida)
Descripción: Sesión inválida, comentario existente
Entradas: creationSession "yyyyy" idComment: 1
Resultado esperado: “ERROR”

PRUEBA UNITARIA 17 (no válida)
Descripción: Sesión válida, comentario inexistente

Entradas: creationSession "xxxxx" idComment: -1
Resultado esperado: “ERROR”

Pruebas de búsquedas

Pruebas unitarias de getCommentGame. Las pruebas unitarias de getTagsGame son similares, aunque cambiando Comment por Tag o comentario por tag.

PRUEBA UNITARIA 18 (válida)
Descripción: Sesión válida, juego existente
Entradas: creationSession "xxxxx" idGame: 1
Resultado esperado: List<Comment>

PRUEBA UNITARIA 19 (no válida)
Descripción: Sesión inválida, juego existente
Entradas: creationSession "yyyyy" idGame: 1
Resultado esperado: null

PRUEBA UNITARIA 20 (no válida)
Descripción: Sesión válida, juego inexistente
Entradas: creationSession "xxxxx" idGame: -1
Resultado esperado: null

Las pruebas de unidad para getCategories son similares aunque sin el parámetro idGame y sin el equivalente al 20, que no existe. Las pruebas de getGames son las siguientes:

PRUEBA UNITARIA 21 (válida)
Descripción: Sesión válida, idFilterer válido, idOrderer válido, fields con campos válidos, values con valores válidos.
Entradas: creationSession “xxxxx” idFilterer: 0, idOrderer: 0, fields: [], values: []
Resultado esperado: List<Game>

PRUEBA UNITARIA 22 (válida)
Descripción: Sesión válida, idFilterer válido, idOrderer válido, fields con campos válidos, values con valores válidos.
Entradas: creationSession “xxxxx” idFilterer: 1, idOrderer: 1, fields: [“idCategory”], values: [“1”]
Resultado esperado: List<Game>

PRUEBA UNITARIA 23 (válida)
Descripción: Sesión válida, idFilterer válido, idOrderer válido, fields con campos válidos, values con valores válidos.
Entradas: creationSession “xxxxx” idFilterer: 2, idOrderer: 2, fields: [“name”], values: [“aaaa”]

Resultado esperado: List<Game>

PRUEBA UNITARIA 24 (válida)
Descripción: Sesión válida, idFilterer válido, idOrderer válido, fields con campos válidos, values con valores válidos.
Entradas: creationSession “xxxxx” idFilterer: 3, idOrderer: 3, fields: [“idGame”], values: [“1”]
Resultado esperado: List<Game>

PRUEBA UNITARIA 25 (no válida)
Descripción: Sesión válida, idFilterer válido, idOrderer válido, fields con campos válidos, values con valores inválidos.
Entradas: creationSession “xxxxx” idFilterer: 3, idOrderer: 4, fields: [“idGame”], values: [“a”]
Resultado esperado: null

PRUEBA UNITARIA 26 (no válida)
Descripción: Sesión válida, idFilterer válido, idOrderer válido, fields con campos inválidos, values con valores válidos.
Entradas: creationSession “xxxxx” idFilterer: 3, idOrderer: 5, fields: [“name”], values: [“1”]
Resultado esperado: null

PRUEBA UNITARIA 27 (no válida)
Descripción: Sesión válida, idFilterer válido, idOrderer inválido, fields con campos válidos, values con valores válidos.
Entradas: creationSession “xxxxx” idFilterer: 2, idOrderer: -1, fields: [“name”], values: [“aaaa”]
Resultado esperado: null

PRUEBA UNITARIA 28 (no válida)
Descripción: Sesión válida, idFilterer inválido, idOrderer válido, fields con campos válidos, values con valores válidos.
Entradas: creationSession “xxxxx” idFilterer: -1, idOrderer: 2, fields: [“name”], values: [“aaaa”]
Resultado esperado: null

Las pruebas unitarias de Jsonp son similares a las descritas aquí aunque añadiendo un valor para el parámetro callback válido.

Construcción

Introducción

En esta fase se documentan aspectos importantes de la implementación del proyecto. Aquí se podrán encontrar las tecnologías y librerías empleadas durante la construcción de las aplicaciones del proyecto y los programas utilizados para su desarrollo. También, se describirán con detalle los problemas encontrados durante la implementación del proyecto, y las decisiones o soluciones tomadas para solventarlos. Por ultimo se informará de los resultados obtenidos de ejecutar las pruebas unitarias descritas en la fase de diseño.

Tecnologías empleadas

Ya se han indicado todas ellas en la fase de diseño. Se pueden distinguir las siguientes tecnologías:

Java:

Lenguaje de programación utilizado para el desarrollo de la parte servidora. Fue seleccionada como el lenguaje de programación a utilizar, por los desarrolladores de la plataforma, de modo que será necesario mantenerla.

PHP:

Lenguaje de programación utilizado para el desarrollo de la parte cliente. Fue seleccionada indirectamente por los desarrolladores de la plataforma como el lenguaje de programación a utilizar puesto que es el lenguaje en que se basa elgg. Igualmente será necesario mantenerlo.

Hibernate:

Librería ORM escrita en Java que ayuda a la implementación de la capa de persistencia eliminando apariciones en el código, de instrucciones SQL y sustituyéndolas, por un lenguaje propio de Hibernate, HQL, que elimina en la medida de lo posible las diferencias entre los distintos tipos de sistemas gestores de base de datos. Como se explicará en el apartado de problemas y soluciones, ha sido mantener la ausencia de consultas SQL.

La librería Hibernate se utiliza junto con anotaciones de la JPA(Java Persistence API), igualmente explicado en el apartado de problemas y soluciones, ha sido posible utilizar en su totalidad únicamente anotaciones JPA, sin utilizar anotaciones propias de Hibernate.

Software de desarrollo.

Algunas de las aplicaciones utilizadas para la realización del proyecto han sido utilizadas antes de la implementación.

Sistemas operativos:

Para el desarrollo del proyecto se ha utilizado la distribución Fedora 16 del sistema operativo Linux en una máquina virtual VirtualBox.

Edición de código:

Para la implementación del código de la parte servidora (lenguaje Java), se utiliza el IDE Eclipse.
Para el desarrollo de la parte cliente, en PHP, se ha utilizado un editor de textos plano y el navegador web Firefox.

Problemas y soluciones.

En este apartado se describirán los principales problemas que el proyectante se ha encontrado durante la implementación tanto de la parte servidora como de la cliente.

Retraso del proyecto y problemas para instalar Kpax en entorno local

El proyecto se ha visto influenciado por varios problemas graves que han retrasado el proyecto e impedido la correcta realización de las fases en sus plazos previstos. Como aclaración el curso comenzó el 29/02/2012.
Estos son:

- El área de PFC de Videojuegos educativos al que pertenece este proyecto ha sido incluido por primera vez este semestre. La asignatura no parecía estar preparada aun al principio del curso puesto que la documentación que describe la arquitectura de la plataforma no se nos entregó hasta el 20/03/2012.
- La tarea de elegir entre realizar un juego o un módulo de kpax estaba prevista para finalizar el 05/03/2012. Pero, no disponíamos de la información suficiente como para hacer esa elección durante la primera semana. La fecha en que me decido a hacer un módulo de kpax es el 10/03. El 17/03 me hacen propuestas de mejoras.
- Junto con otro compañero, hemos sido los dos primeros alumnos que han tratado de realizar mejoras en la plataforma Kpax y en consecuencia no existía tampoco documentación previa de otros alumnos ni proyectos en los que basarse.
- Recibimos las instrucciones para instalar la plataforma el 12/04/2012. Hasta entonces disponíamos de unas instrucciones muy vagas sobre cómo hacerlo, con las que era absolutamente imposible conseguir instalarlo correctamente.

  A pesar de ello las instrucciones estaban incompletas puesto que a pesar de que traté de instalar la plataforma en tres equipos diferentes con sistemas operativos diferentes, la parte elgg no realizaba correctamente llamadas a la parte servidora y por eso no podía comprobar el funcionamiento real de la aplicación durante la realización de las fases de análisis ni diseño. Esta es la razón principal de que perdiera cerca de otro mes.

Con todo, consigo finalmente instalar correctamente la plataforma en un entorno local el 17/06/2012 a las 19:20 horas -los pasos faltantes en las instrucciones de instalación las describiré a continuación-. La fecha de entrega del proyecto estaba prevista para el 19/06 (dos días después), aunque con todos los problemas que había tenido, los tutores me concedieron hasta el 26/06 como fecha límite de entrega del proyecto. En resumen, disponía de aproximadamente 9 días para realizar la fase de implementación y terminar la documentación.

Los pasos que tuve que realizar para poder terminar de instalar correctamente en un entorno local la plataforma, y que en mi opinión faltan en las instrucciones de instalación actualmente, son:

1. En las instrucciones en https://github.com/jsanchezramos/mods-kpax actualmente se indica que de las dos claves públicas/privadas generadas con la API de administración es siempre la pública, dando a entender como que elgg utiliza internamente la clave privada y que tanto en */www/elgg/mods/kpax/lib/kpaxSrv.php* como en

/src/main/java/uoc/edu/svrKpax/util/ConstantsKPAX.java se debe introducir la pública. Para el caso del proyecto esto no ha sido así y se ha tenido que utilizar la privada en el fichero ConstantsKPAX.java.

2. En el fichero kpaxSrv.php anterior, se tuvo que cambiar el valor de la variable $url para que apuntara al servidor JBoss en local que recibe las peticiones HTTP en el puerto 8080, es decir que se tuvo que cambiar por *$url = "http://localhost:8080/webapps/srvKpax"*.
3. En el mismo directorio que kpaxSrv.php, se tuvo que cambiar en el fichero kpaxOauth.php las variable URL y API_URL por las rutas al servidor elgg y de JBoss:
   - URL = 'http://localhost/elgg/';
   - API_URL = 'http://localhost:8080/webapps/srvKpax';
4. En la base de datos de kpax ha sido necesario insertar un usuario con el nombre del usuario administrador de elgg (admin) y un valor para el campo secret, en este caso se ha utilizado como valor la contraseña del usuario aunque no parece ser necesariamente ese valor. También se ha necesitado insertar un Realm donde el alias ha sido el nombre del usuario (admin) y relacionar ambas tuplas en la tabla UserRealm, por medio de sus identificadores
5. Además aunque se indica en el manual de instalación de los módulos de elgg, conviene indicar también en el manual de instalación de kpax que el término a utilizar para generar las claves pública y privada es *kpax*.

Con un margen de 9 días (7 en realidad, puesto que debía terminar la documentación de esta fase), decidí realizar una entrega lo más cerrada posible, cuyas modificaciones hubiesen sido probadas y funcionasen correctamente, más que tratar de realizar todos los cambios propuestos y que estos no funcionasen correctamente.

Como hasta el momento sólo había podido codificar los servicios basándome en el código existente y según mi diseño, en un principio decidí entregar únicamente la implementación de los servicios web. Por otro lado, la plataforma parece haber sido bien diseñada, y los cambios no requieren de mucho tiempo de implementación. Finalmente, he conseguido incluir el uso de los servicios web en elgg a costa de eliminar otras funcionalidades. De forma resumida, las funcionalidades más importantes eliminadas son:

- No se incluye la posibilidad de añadir, ni eliminar una categoría ni consultar su ficha. Las categorías deben ser añadidas en la tabla Category directamente.
- En la vista de listado de juegos no se incluyen los campos para poder filtrar y/u ordenar el listado. Aunque sí que se realiza la llamada al servicio web de getGames desde los ficheros PHP, por tanto se realizará la búsqueda según los valores por defecto (idOrderer = 0 e idFilterer = 0, es decir, sin realizar ningún filtrado ni ordenación) tanto en el listado de todos los juegos, como los propios como en los de juegos de amigos.
- No se han podido realizar los cambios de aspecto esperados ni en la vista del listado de juegos (por no añadirse los nuevos campos) ni en la ficha de un juego. Es decir, el aspecto descrito en la fase de prototipado de la fase de análisis. Esta funcionalidad es la que requeriría más tiempo.
- No se han añadido los campos de categoría ni fecha de creación en elgg para la ficha de un juego y sólo las muestra si consigue obtener la información del juego correctamente llamando al servicio web de getGame y de getCategory. Esto implica que a la hora de añadir un juego sea necesario indicar el identificador de la categoría en vez de elegir uno de entre una lista desplegable.
- Otras eliminaciones menores que se explicarán en los apartados siguientes.

Modificación en la table Comment:

En la fase de diseño se indicó que la tabla Comment tendría dos campos con el texto del mensaje y

la fecha del comentario. Sin embargo, las dificultades encontradas para poder instalar la plataforma en local (explicadas en el apartado anterior) y poder estudiar su funcionamiento han provocado un retraso muy grande del proyecto haciendo peligrar su finalización a tiempo antes de la fecha límite. Esto ha propiciado que se eliminen todos los cambios que no son estrictamente necesarias. Como no se utiliza el texto del mensaje ni la fecha del comentario para la realización de búsquedas, finalmente no se han introducido.

No creación de índice en GameView

En la fase de diseño se indicó que para acelerar las consultas de listado de juegos se debería crear un índice sobre las columnas popularity, name e idGame. Sin embargo hubo algo que el proyectante no tuvo en cuenta: En vistas no es posible generar índices. No obstante, con los índices primarios y secundarios se consigue igualmente un buen rendimiento.

Escasa documentación sobre el uso de JAX-RS:

JAX-RS: Java API for RESTful Web Services es una API Java que provee soporte a la creación de servicios web acordes con el estilo arquitectónico Representational State Transfer, o REST, utilizando anotaciones, lo que simplifica el desarrollo y despliegue de clientes web service.

Es sobre esta tecnología sobre la que he tenido problemas para encontrar información, debido a que es relativamente novedoso y su actual popularidad aun no ha propiciado la generación de documentación por Internet, tal y como puede haber para otras APIs como por ejemplo Hibernate. No obstante, al final he terminado encontrado respuestas a mis dudas personales acerca de esta API.

Anotaciones utilizadas.

Las clases Games y Jsonp utilizan anotaciones JAX-RS para publicar los servicios web.
Cada servicio web es una función pública. Todo servicio debe tener al menos dos anotaciones. Toda anotación se deberá indicar justo antes de la declaración de la función. Estas dos funciones son:

- @GET o @POST: indica si la información se recibe por medio de una petición HTTP GET o POST. Por ejemplo, como la función getGames ahora recibe un array de datos, que en la teoría puede ser de tamaño infinito, esta función ha sido declarada por medio de @POST.
- @Path ("URL"): Se declara una anotación @Path antes de la declaración de la clase que define la URL donde se encuentran todos los servicios web de la clase y otro @Path justo antes de cada función que indica el resto de URL donde se publica cada servicio web específico.

JAX-RS permite la sobrecarga de funciones siempre que @Path de las funciones no apunte a la misma URL. Por esta razón he tenido que sobrescribir la función actual de addGame. Por precaución también lo he hecho con getGames aunque no es estrictamente necesario por no estar publicadas en la misma URL y además no parece usarse en la parte cliente aún.

Cuando esto ocurre se lanza una excepción al desplegar el proyecto en JBOSS como el que sigue: *Producing media type conflict. The resource methods […] and […] can produce the same media type*. Donde […] son las cabeceras de las funciones concretas que utilizan la misma URL.

La anotación *@Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})* se utiliza para poder devolver objetos no primitivos como resultado de la función, tales como Game o List<Comment>. Igualmente cuando los parámetros contienen algún tipo de dato no primitivo, se debe indicar la anotación *@Consumes({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})*. En la clase Jsonp, por servir para realizar llamadas con AJAX, debe devolver el resultado como javascript, por ello, la anotación es *@Produces("application/x-*

javascript")

En la declaración de las cabeceras de funciones, hace falta declarar por cada parámetro, su valor está contenido en la URL, entre los parámetros recibidos por GET o si por el contrario se reciben en los parámetros POST. Por tanto cada uno de los parámetros de la función deben ir seguidos de una anotación de cualquiera de los tipos siguientes:

1. @PathParam: El valor es recibido como parte de la URL definida en la anotación @Path y cada nombre de parámetro viene definido entre llaves. Por ejemplo, si el valor de @Path es *@Path("/{param}/get/{id}")*; la cabecera de la función deberá contener dos parámetros del tipo primitivo esperado: *public Game getGame (@PathParam("id") int idGame @PathParam("param") String secretSession)*
2. @QueryParam: El valor es recibido como un valor GET de la llamada HTML realizada. Por ejemplo, si se hace la llamada /game/get?param=2ab40c&id=1 la cabecera de la función deberá contener dos parámetros del tipo primitivo esperado: *public Game getGame (@QueryParam("id") int idGame @QueryParam("param") String secretSession)*
3. @FormParam: El valor es recibido como un valor POST de la llamada HTML realizada. Por ejemplo, si se envían los parámetros param=2ab40c&id=1 por POST, la cabecera de la función deberá contener dos parámetros del tipo primitivo esperado: *public Game getGame (@FormParam("id") int idGame @FormParam("param") String secretSession)*

Hubo que tener especial cuidado con la función getGames, porque JAX-RS no admite recibir arrays de datos primitivos, porque el tipo de dato que se utiliza por defecto para este propósito no es un array sino un objeto de la clase List cuya generalización sea la clase envolvente de la clase primitiva esperada.

Así, como se quería recibir un array de Strings (String []), se tuvieron que cambiar los parámetros por List<String>. Pero si, por ejemplo, hubiese querido recibir un array de longs, habría tenido que cambiarlo por List<Long>. En resumen, la cabecera de la función getGames cambio a: *public List<Games>(String campusSession, int idOrderer, int idFilterer, List<String> fields, List<String> values)*; No obstante, este cambio de función no supone mayor problema y la interacción con la misma sigue siendo similar.

A continuación se muestran dos de los ejemplos reales implementados, más completos. El primer ejemplo se encuentra en la clase Games.java y el segundo en la clase Jsonp.java:

```
@POST
@Path("/{param}/list/{idOrderer}/{idFilterer}")
@Consumes({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
@Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
public List<Game> getGames(@PathParam("param") String campusSession,
@PathParam("idOrderer") int idOrderer, @PathParam("idFilterer") int idFilterer,
@FormParam("fields") List<String> fields, @FormParam("values") List<String> values) {
        return gBo.listGames(campusSession, idOrderer, idFilterer, fields, values);
}
```

```
@POST
@Path("/{param}/list/{idOrderer}/{idFilterer}")
@Consumes({MediaType.APPLICATION_JSON})
@Produces("application/x-javascript")
public JSONWithPadding getGamesJsonp(@PathParam("param") String campusSession,
@PathParam("idOrderer") int idOrderer, @PathParam("idFilterer") int idFilterer,
@FormParam("fields") List<String> fields, @FormParam("values") List<String> values,
@QueryParam("jsoncallback") String callback) {
```

```
        //return gBo.listGames(campusSession);
        return new JSONWithPadding(gBo.listGames(campusSession, idOrderer, idFilterer, fields,
values), callback);
}
```

Modificaciones y nuevas funciones añadidas.

Durante la fase de diseño no se tuvieron en cuenta algunas consideraciones que han propiciado la modificación de funciones Java o la creación de nuevas. Algunas provocadas por la inexperiencia del proyectante con JAX-RS o con el funcionamiento de elgg, como la modificación de los parámetros String[] por List<String> en la función getGames, explicado en el apartado anterior, y otras por no haber podido instalar correctamente la plataforma en local hasta un fecha muy cercana al límite de entrega del proyecto.

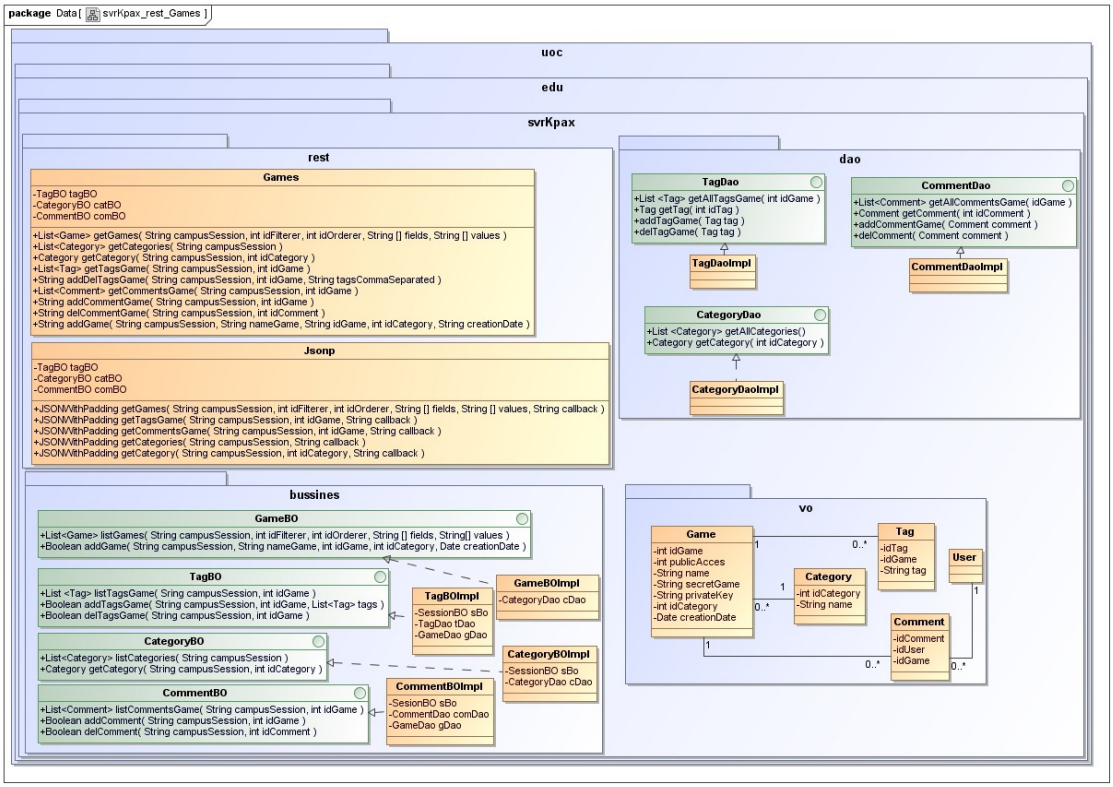
Los cambios son los siguientes:

1. La modificación de los parámetros String[] por List<String> en la función getGames de las clases Games y Jsonp del paquete uoc.edu.svrKpax.rest, explicado en el apartado anterior.
2. La creación de una nueva función getCategory en la clase Games y Jsonp del paquete uoc.edu.svrKpax.rest que obtuviera un objeto Category con la información de una categoría a partir de su identificador idCategory.

   Durante la fase de diseño se esperaba que desde la ficha de adición de un juego se mostrara un listado desplegable que obtuviera la lista completa de las categorías a mostrar a través de la función getCategories de la clase Games o de la clase Jsonp. En elgg se guardaría el nombre y el código de la categoría para que al consultar la ficha de un juego se mostrara el nombre directamente sin realizar ninguna llamada extra.

   Pero, debido al poco tiempo del que se disponía para la fase de implementación, no he podido invertir más en buscar la manera de mostrar un listado desplegable con elgg. Además, también por falta de tiempo se ha decidido no añadir los ficheros de alta, baja y consulta de categorías con el fin de poder realizar una entrega más o menos cerrada.

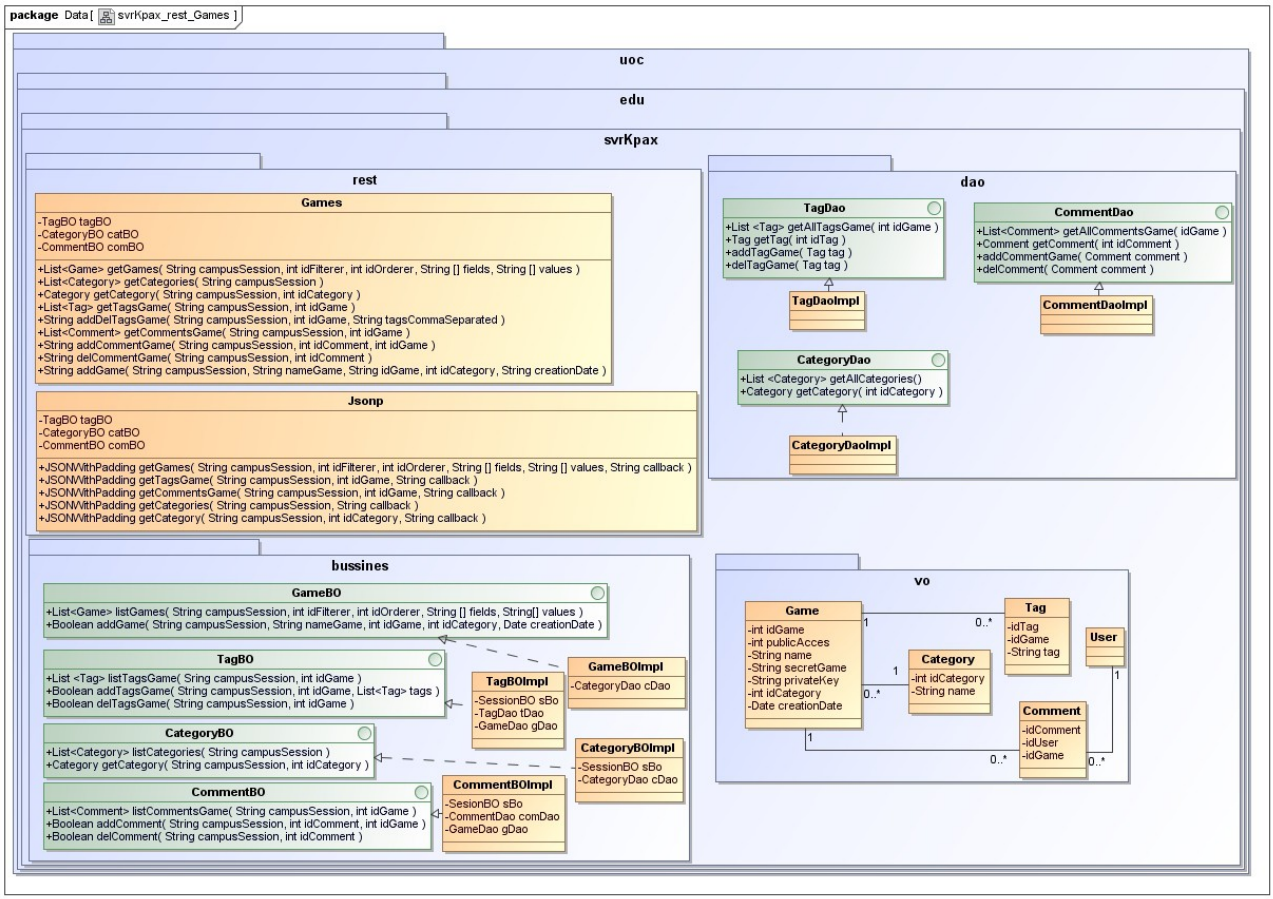
   Hacer esta pequeña adición (una función más en la clase Games y en Jsonp y otra en CategoryBO y en su implementación) requería menos tiempo que añadir los campos necesarios a la ficha de creación de un juego y la creación de cuatro ficheros PHP nuevos. A continuación se muestra un diagrama de clases con las nuevas funciones:



En consecuencia el modo de funcionamiento es el siguiente. Las categorías son añadidas en la tabla Category de la base de datos de kpax de forma directa, junto con su nombre. Durante la creación de un juego se solicita el identificador de las categorías añadidas a la base de datos. A la hora de consultar la ficha de un juego, se realizará la consulta del nombre de la categoría por medio de la función getCategory creada.

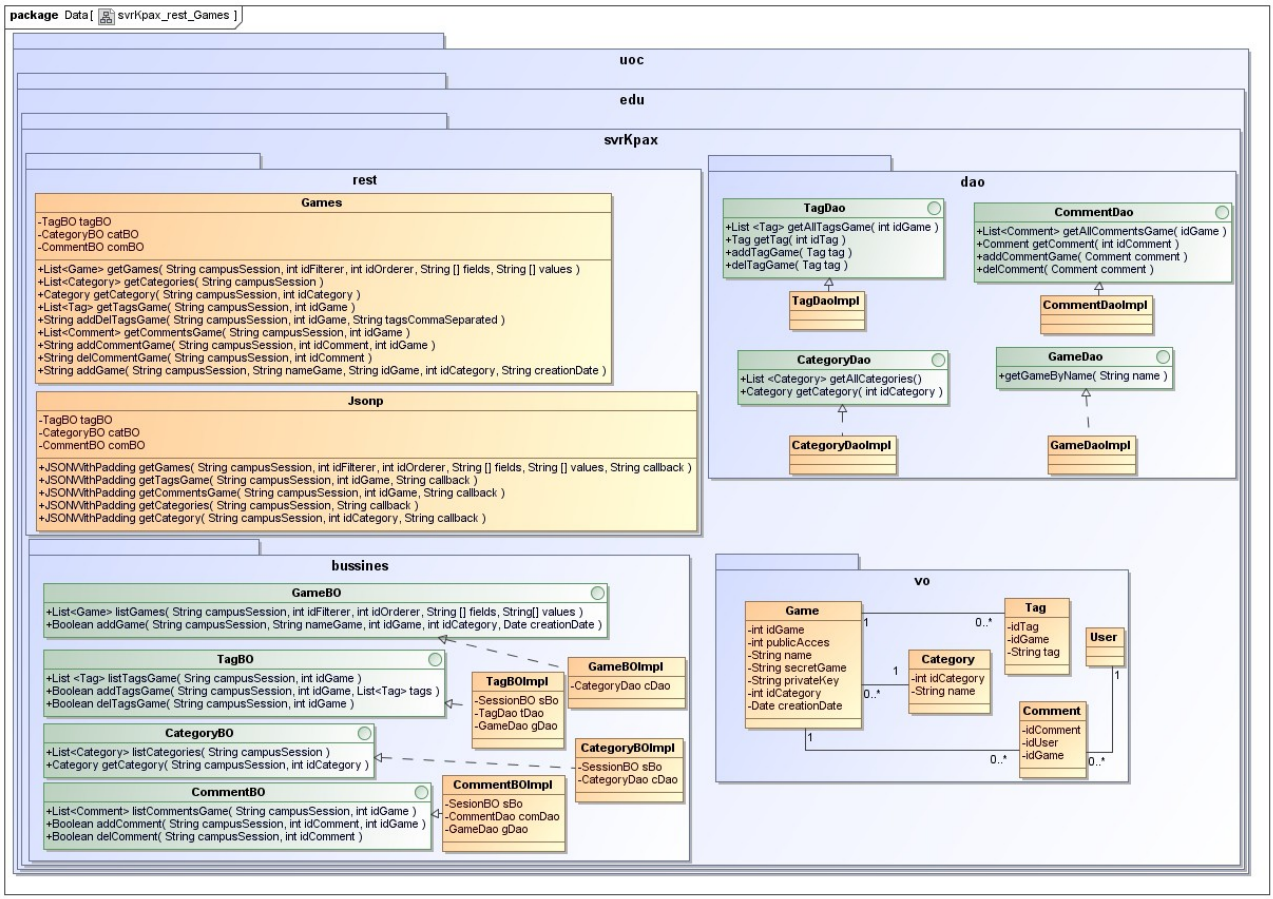
3. Una vez más por falta de conocimiento del funcionamiento de elgg y debido a que la plataforma no me funcionó correctamente hasta casi el final del curso, por no poder fijarme en el funcionamiento real, durante la fase de diseño me basaba en el diseño referente a la entidad Game y al de la base de datos para diseñar la entidad Comment. Por eso, creía que el identificador de una entidad, por ejemplo un juego, se añadía de forma incremental dejando el valor idGame vacío y que este parámetro en la función addGames de la clase Games aparecía como un parámetro opcional.

   La realidad ha resultado ser diferente, puesto que aunque el identificador de la tabla Games sea incremental, en una situación normal siempre se establece un valor a idGame. Este valor es el identificador de una entidad elgg, el cual es también un número. Por esta razón ha sido necesario añadir un campo idComment a la función addComment de la clase Games y en consecuencia a la de addComment de CommentBO. El siguiente diagrama de clases siguiente muestra el cambio:



4. Durante la construcción del proyecto, una vez conseguí hacer funcionar correctamente la plataforma en local, observé que en elgg no era posible añadir dos juegos con un mismo nombre, sin embargo en la función addGame actual no existía ninguna comprobación para determinar si existía ya algún juego repetido.

   Aprovechando que debía reimplementar entera dicha función, añadí en la clase GameBOImpl, en GameDao y en su implementación una nueva función getGameByName que devuelve el juego (objeto de la clase Game) con el nombre name. El siguiente diagrama de clases siguiente muestra el cambio:



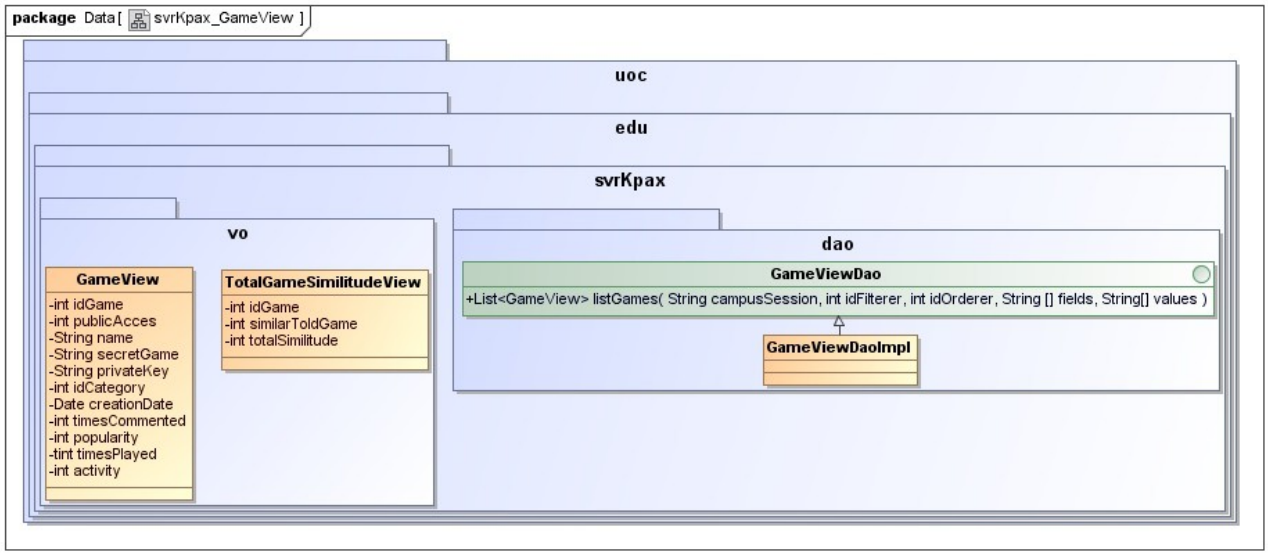
5. Durante la fase de diseño el proyectante decidió añadir a la clase GameView los juegos Game similares a otro juego concreto (al que hace referencia el campo idGame de GameView) guardándolos en una variable denominada similarTo. Para ello se pretendía crear una vista donde cada tupla contuviese todos los campos de ambos juegos, el original y el similar, junto con su puntuación total de similitud.

   De este modo cada tupla para un idGame original concreto se correspondería con un objeto Game de la lista similarTo, mientras que la información del juego original en cualquiera de esa tupla se podría utilizar para obtener la información del juego original.

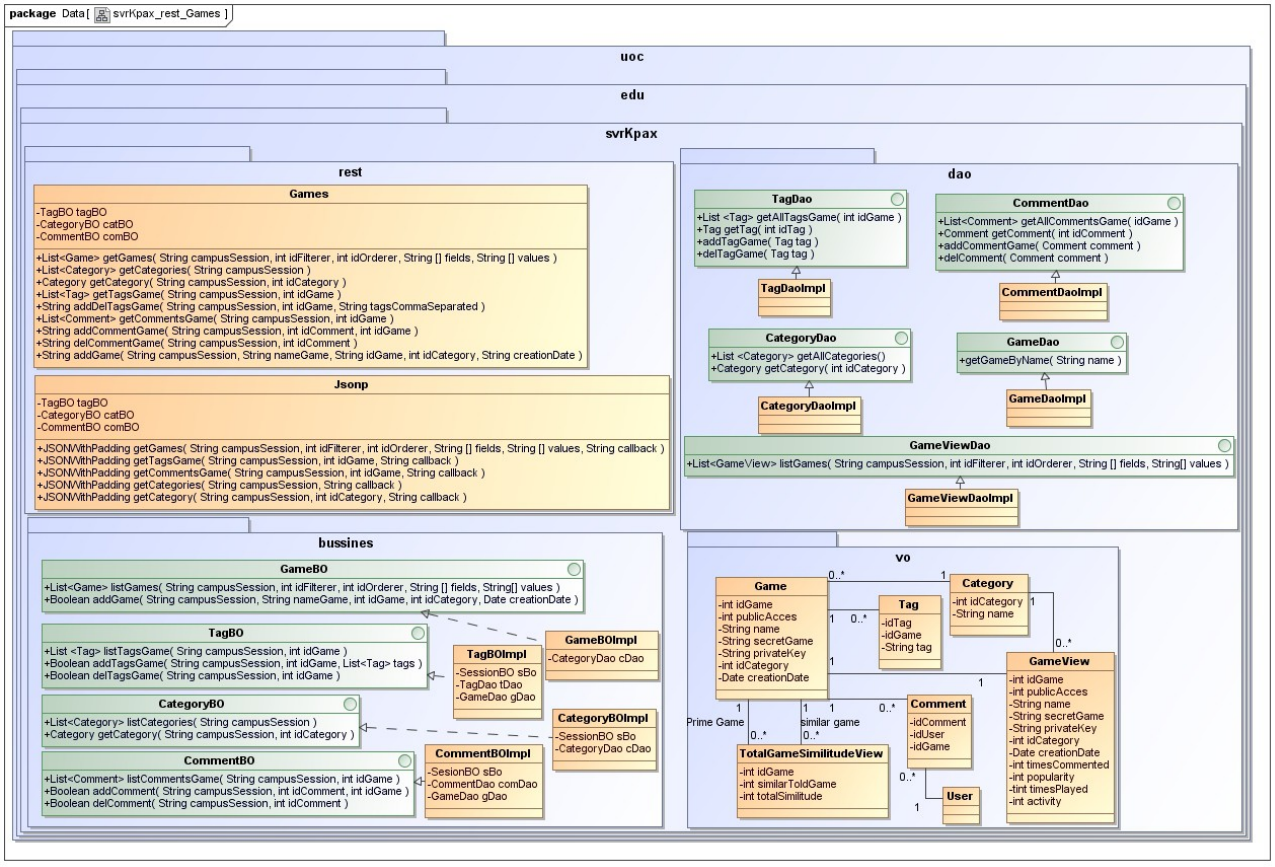
   En la práctica esto no ha sido posible, y se han necesitado dos vistas más, una que devuelve los identificadores de juegos original y similar junto con la puntuación total de similitud (TotalGameSimilitudeView) y una vista intermedia para poder construir esta primera (GameSimilitudeView) que contiene los identificadores de juego original, id del juego similar y el valor de tag o categoría para el que coinciden.

   La solución, entonces, ha sido crear una clase nueva que encapsula la información de la vista TotalGameSimilitudeView. Por ello se ha denominado de la misma forma. Esta clase es la que se utiliza finalmente en la función getGames de la clase GameDaoImpl durante la construcción de la consulta HQL a realizar. Como la clase creada no se utiliza en más sitios no ha sido necesaria la creación de su clase Dao. Tarea que se habría realizado si se hubiese dispuesto de tiempo suficiente.

   Además, como no se están utilizando anotaciones OneToOne, ManyToOne, OneToMany ni ManyToMany, el diseño inicial para GameView no ha sido posible implementarlo. Ha sido necesario duplicar todos los campos de la clase Game en GameView. El siguiente diagrama de clases siguiente muestra el cambio:



En resumen, el diagrama de clases final es el siguiente:



Fichero Hibernate.cfg.xml

El fichero Hibernate.cfg.xml contiene los datos de acceso a la BD, los datos de configuración del pool de conexiones a la BD y muchas más propiedades para configurar las conexiones que realiza Hibernate. No se han realizado cambios en este fichero y los comentarios incluidos explican el contenido. No obstante, indicaré algunas modificaciones que se podrían incluir, aunque estas no han sido probadas en este proyecto (sí en un proyecto anterior del proyectante):

1. Si se quisiera configurar un pool de conexiones (mejora propuesta en la fase de diseño):

```
<property name="c3p0.min_size">1</property> //Conexiones mínimas
<property name="c3p0.max_size">20</property> //Conexiones máximas
```

```
<property name="c3p0.timeout">3000</property> //Expresado en segundos
<property name="c3p0.max_statements">50</property> //Núm. máx. Sentencias
```

2. Si se quisieran configurar las transacciones (mejora propuesta en la fase de diseño).
```
<property name="Hibernate.connection.autocommit">false</property>
      //Habilita o deshabilita el autocommit.
<property name="Hibernate.connection.isolation">2</property> //Establece
      el nivel de aislamiento de la transaccion. Sus valores son: 1
      UNCOMMITED, 2 READ COMMITED, 3 SERIALIZABLE, 4 REPETEABLE READ.
```

3. Para mostrar las instrucciones que ejecuta Hibernate por la salida estándar. Útil en entornos de desarrollo, aunque algunas ya están incluidas. En entornos de producción es conveniente quitarlos para aligerar las consultas:
```
<property name="show_sql">true</property> //Muestra las instrucciones SQL
por la salida estándar.
<property name="format_sql">${devMode}</property> //Formatea las
instrucciones SQL. Por ejemplo si se indica true no se muestran todo
seguido.
<property name="use_sql_comments">true</property>//Incluye comentarios
antes de cada instrucción SQL, como por ejemplo la clase desde la que se
ejecuta la instrucción.
```

Principales anotaciones JPA/Hibernate:

Como ya se ha dicho para representar la estructura de la BD en clases, se utilizan las anotaciones de la JPA. Las principales anotaciones son:

- @XmlRootElement justo antes de la declaración de la clase, para indicar que dicha clase se represente como un elemento XML en un documento XML. Esto permite enviar objetos de la clase como salida de los servicios web siempre que se utilice la anotación @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML}), explicada anteriormente. Para hacer esto, además la clase debe implementar la interfaz Serializable que permite enviar objetos no primitivos por la red.
- @Entity justo antes de la declaración de la clase, para indicar que dicha clase representa una entidad de un modelo de entidad relación. Véase ejemplo siguiente.
- @Table justo antes de la declaración de una clase, para indicar que la clase representa una tabla de la BD cuyo nombre es el mismo que el declarado en el parámetro name. Ejemplo:

```
@Entity
@Table(name = "Game")
public class Game {
```

El parámetro uniqueConstraints a continuación del de name indica un array de anotaciones @UniqueConstraint cuyo parámetro columnNames indica el nombre de las columnas que forman el índice. Ejemplo:

```
@Entity
@Table(name = "Category",
            uniqueConstraints =
            {
                  @UniqueConstraint(columnNames= {"name"})
            }
      )
public class Category
{
```

- @Column justo antes de la declaración de una variable para indicar que dicha variable

representa una columna cuyo nombre sea el mismo que el valor del parámetro name. También es posible indicar la anulabilidad de la columna mediante el parámetro nullable, o la longitud máxima que pueden tomar los valores de la columna con el parámetro length o el tipo de dato que debe tomar en la BD con el parámetro columnDefinition entre otros.

Por ejemplo con las clases Game y GameView he tenido que indicar que los campos creationDate e idCategory sean anulables para que al pasar a producción no falle si se intenta leer un juego al que aun no se le han añadido dichos valores. Aunque en casos normales esto nunca deba darse.

```
public class Game{
	[...]
	@Column(name = "creationDate", nullable = true)
	public Date getCreationDate(){
		return creationDate;
	}
	[...]
```

- @Id junto con @Column justo antes de la declaración de una variable o de una función get para indicar que la columna a la que representa dicha variable forma el índice primario de la tabla. Ejemplo:

```
public class Local {
	[...]
	@Id
	@Column(name = "idTag")
	public int getIdTag(){
		return idTag;
	}
	[...]
```

Eliminaciones en cascada:

Como no se han utilizado anotaciones @OneToOne, @ManyToOne, @OneToMany o @ManyToMany en las clases, porque supondría tener que cambiar y probar todas las clases del paquete vo, no se puede utilizar el parámetro cascade que delegaría el trabajo de controlar las eliminaciones en cascada en Hibernate.

Por ello, debe hacerse de una forma manual. Esto es, comprobar durante la eliminación de una entidad si existen elementos de otras entidades que dependan de la primera, para después eliminarlas. Por ejemplo, si se elimina un juego, elgg se encargará de eliminar también todos los comentarios o tags que dependan del juego en la base de datos de elgg, pero eso también debe hacerse en la base de datos de kpax, para lo que habrá que modificar la implementación de la función Java delGame.

Por desgracia, implementar las eliminaciones en cascada de esta manera, ha sido otra de las tareas que el proyectante ha tenido que dejar pendientes por falta de tiempo.

Principales problemas y cambios en la parte elgg:

Como se indica en la fase de diseño, cada una de las funciones debían añadirse una a una en el fichero kpaxSrv.php. La clase dispone de la función service que permite realizar una llamada vía GET o POST, a una URL concreta indicando los parámetros a enviar.
Dependiendo de si el servicio recibe la petición por POST o por GET, se deben incluir las llamadas

siguientes:

- GET: this->service($url);
- POST: $this->service($url, “POST”, $body);

Donde $url es una url que concuerda con el solicitado en la anotación @Path -los valores incluidos son recogidos por los parámetros con @PathParam- de la clase Games o Jsonp y $body sigue las reglas HTML para enviar parámetros por Get o Post -con Get, el primer parámetro debe venir seguido de un símbolo de interrogación final '?' y el resto de parámetros del ampersand '&', por ejemplo: ?field=name&values=a; y en el POST el inicial no viene seguido de nada: field=name&values=a -.

Además si el servicio web devuelve un objeto en formato XML, es decir, en aquellos en los que se incluía la anotación *@Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})*, que se correspondía con los no primitivos, la llamada a service debía encapsularse en la función json_decode, que permitía decodificar el XML recibido desde JSON en un array cuyos valores son los campos de la clase devuelta.

Además me encontré con un problema al basarme en las funciones presentes. Algunas de ellas no tenían la clausula return, por ejemplo addLikeGame o delLifeGame, de modo que al copiar de estas tampoco las ponía en las mías. Esto causaba el siguiente efecto cuando trataba de mostrar un mensaje de error al ejecutar la función dependiendo de si el servicio fallaba o no: Siempre se mostraba mensaje de error.

Por último para hacerla función de getGames, me base en la actual. En esta función, la actual, se devolvía el valor por medio de la función var_dump, que copié pensando que se utilizaría por alguna razón en especial, que desconocía. El resultado fue que al ejecutarlo en elgg se mostraba el array de juegos como una cadena y su contenido no era accesible. Es decir, es como si se ejecutara una función echo de PHP con el contenido recuperado del servicio web. Como solución utilicé igualmente la función json_decode.

A continuación se muestran dos ejemplos de funciones. La primera GET y la segunda POST, utilizando las funciones declaradas.

```
public function getTagsGame($campusSession, $idGame) {
	$listTags = json_decode($this->service("game/tag/" . $campusSession . "/list/" .
$idGame));
	return $listTags;
}
```

```
public function addDelTagsGame($campusSession, $idGame, $tagsCommaSeparated) {
	$body = 'secretSession=' . $campusSession . '&tags=' . $tagsCommaSeparated;
	return $this->service("game/tag/" . $idGame . "/addDel", "POST", $body);
}
```

Utilizar las funciones de esta clase es tan sencillo como declarar una variable de la clase y ejecutar sus funciones:

```
$objKpax = new kpaxSrv(elgg_get_logged_in_user_entity()->username)
```

Esto es posible realizarlo en una línea porque la clase kpaxSrv se importa a elgg desde el fichero start.php de la raíz del módulo. En este fichero se añaden las librerías de la carpeta lib o las acciones de la carpeta actions entre otras cosas y permite a elgg conocer la ruta de todos los ficheros PHP que incluye el módulo. En caso de haber implementado la posibilidad de añadir, eliminar y consultar la ficha de una categoría habría necesitado probablemente añadir un módulo nuevo que incluiría un fichero de este tipo.

Elgg pone a disposición del desarrollador una diversa cantidad de funciones que permiten interaccionar con elgg y su base de datos. Las funciones utilizadas más importantes son las siguientes:

- register_error(elgg_echo("mensaje de error")): Muestra un mensaje de error en color rojo por medio de un div en la esquina superior derecha del navegador tras terminar la carga de la pantalla. Este mensaje se desvanece al cabo de un tiempo. Se muestra un mensaje de error por cada register_error ejecutado.
- system_message(elgg_echo("mensaje")): Muestra un mensaje informativo en color negro por medio de un div en la esquina superior derecha del navegador tras terminar la carga de la pantalla. Este mensaje se desvanece al cabo de un tiempo. El mensaje no se muestra si se ha registrado algún mensaje de error.
- elgg_get_logged_in_user_entity(): recupera un objeto de la clase ElggEntity que contienen la información del usuario. Su variable username contienen el nombre del usuario. Dato requerido para construir un objeto de la clase kpaxSrv.
- elgg_list_entities: Esta es la función estrella, utilizada en los ficheros all.php, owner.php y friends.php de la carpeta elgg/mod/kpax/pages/kpax/ para listar en elgg de forma formateada (tal y como se hace actualmente) los juegos recuperados con el servicio getGames.

  Requiere de un array con las opciones de búsqueda que permite filtrar, ordenar y paginar el conjunto de entidades a listar. El conjunto de opciones por defecto utilizado para listar los juegos en los tres ficheros es el utilizado en el fichero all.php hasta el momento:

```
$options = array(
   'types' => 'object',
   'subtypes' => 'kpax',
   'limit' => 10,
   'full_view' => false,
      );
```

    - El campo 'types' y 'subtypes' restringe la búsqueda a los objetos de subtipo kpax, es decir, los juegos.
    - Con el campo 'limit' se establece un límite de 10 juegos por página.
    - La opción 'full_view' indica que no se muestren ni la descripción, ni la categoría ni la fecha de creación del juego, que son mostradas con full_view = true. Los campos a mostrar con full_view = false y full view = true se definen en el fichero elgg/kpax/views/default/object/kpax.php. Este fichero es el que se tendría que haber modificado si se hubiera incluido el cambio de aspecto previsto en el apartado de prototipado en la fase de análisis.
    - Además, para poder mantener el filtrado de juegos con los mismos juegos que se obtienen con el servicio web getGames, he tenido que utilizar en todas las vistas (all.php, owner.php y friends.php) el campo 'guids' con un array con los identificadores de los juegos devueltos.
    - Por último, mantener el orden recuperado con el servicio web ha sido lo más problemático en estas vistas. El problema es que la función lista, por defecto, los juegos ordenados por la fecha de creación de la entidad (en elgg). Esto anulaba la ordenación solicitada al servicio web. De modo que he tenido que utilizar el campo 'order_by' que requiere un valor siguiendo la sintaxis SQL en combinación con clausulas CASE … WHEN... THEN, dando un valor mayor a cada identificador obtenido desde el servicio web de forma secuencial para luego ordenar por este valor.
    - El listado de juegos propios y de juegos amigos requerían además de otros dos campos:
        - El de juegos propios requería el campo 'container_guid' indicando como valor el

identificador del usuario propietario.

- Para el listado de juegos de amigos se ha utilizado el campo 'owner_guids' indicando los identificadores de los usuarios propietarios. Estos valores se obtienen por medio de la función elgg get_user_friends_of, que devuelve un array de objetos de la clase ElggUser hasta el limite establecido como parámetro (999999).

  Aunque se ha establecido un límite grande, si en alguna ocasión se introducen los amigos en la base de datos de kpax, es conveniente modificar esta función por alguna llamada a algún servicio web que devuelva todos los identificadores de usuarios amigos, sin establecer ningún límite.

- get_subtype_from_id: Indicando como parámetro el identificador del subtipo de una entidad ElggEntity la función devuelve la denominación del subtipo de la entidad. Esta función se ha utilizado en el fichero add.php y delete.php del directorio elgg/actions/comments, para determinar si el comentario a añadir/eliminar es sobre un juego, de subtipo kpax. Así se evita intentar añadir/eliminar un comentario en la base de datos de kpax cuando el comentario no está asociado realmente a ningún juego.

Resultados pruebas unitarias.

El resultado de correr las pruebas unitarias ha sido exitoso en todos los casos. Se pueden consultar los datos a introducir en las pruebas y el resultado que cabe esperar en el apartado de “diseño pruebas” de la fase de diseño.

Por otro lado, durante la creación de las funciones se detectaron tres situaciones adicionales para la función getGames que no se tuvieron en cuenta durante la fase de diseño o que como en caso de las pruebas unitarias 29 y 31 dependen de la implementación. En consecuencia, estas son pruebas unitarias de caja blanca. En resumen, las pruebas añadidas son las siguientes:

PRUEBA UNITARIA 29 (válida)
Descripción: Sesión válida, idFilterer válido que requiera campos field, idOrderer válido, fields con campos válidos y repetidos, values con valores válidos.
Entradas: creationSession “xxxxx” idFilterer: 2, idOrderer: 0, fields: [“name”, “name”], values: [“a”, “b”]
Resultado esperado: List<Game>. Los valores de campos repetidos son sobrescritos. Por ello, se filtra por los juegos con nombre “b”.

PRUEBA UNITARIA 30 (válida)
Descripción: Sesión válida, idFilterer válido, idOrderer inválido, fields con varios campos, values con valores válidos.
Entradas: creationSession “xxxxx” idFilterer: 2, idOrderer: -1, fields: [“name”], values: [“aaaa”]
Resultado esperado: List<Game>. Se espera que aunque no se utilicen, se permita recibir más de un campo field y más de un valor.

PRUEBA UNITARIA 31 (no válida)
Descripción: Sesión válida, idFilterer inválido, idOrderer válido, fields con campos válidos, values con valores válidos. No existen juegos o el resultado filtra todos.
Entradas: creationSession “xxxxx” idFilterer: 0, idOrderer: 2, fields: [“name”], values: [“1”]

Resultado esperado: List<Game>

En las pruebas realizadas por el proyectante el valor de sesión válido no era 'xxxxx' sino '63036cd97ef7a50387a0be48decf873181819b12', un identificador de juego válido era el 67 en vez del 1, con el nombre 'a' en vez de 'aaaa'.

Las pruebas fueron realizadas por medio de una función PHP situada en el fichero kpaxSrv.php. Esta función, que se muestra a continuación se ejecutaba desde el constructor de la clase (public function __construct). De este modo cada vez que se listaban los juegos se ejecutaba. Este modo de probar tenía una pega, la función test se ejecutaba dos veces durante la carga de la página. Por otro lado, ejecutar las pruebas a mano era más rápido que hacerlo con JUnit (opción prevista hasta que se comenzó con la fase de programación) porque requería aprender a utilizar la librería y no quedaba tiempo para ello (ya se ha explicado porqué).

La función test, es la siguiente. Esta función y su llamada desde el constructor se han eliminado en la entrega.

```
private function test()
{
	//PRUEBAS ADDDELTAGGAME
	// Prueba 1
	$listTags = $this->addDelTagsGame("63036cd97ef7a50387a0be48decf873181819b12",
66, "a, b,c");
	// Prueba 14
	$listTags = $this->addDelTagsGame("63036cd97ef7a50387a0be48decf873181819b12",
66, "a, y,z");
	// Prueba 2
	$listTags = $this->addDelTagsGame("yyyyy", 1, "x, y,z");
	// Prueba 3
	$listTags = $this->addDelTagsGame("63036cd97ef7a50387a0be48decf873181819b12", -1,
"x, y,z");
	// Prueba 4
	$listTags = $this->addDelTagsGame("63036cd97ef7a50387a0be48decf873181819b12",
66, "");

	//PRUEBAS ADDCOMMENTGAME
	// Prueba 1B
	$listTags = $this->addCommentGame("63036cd97ef7a50387a0be48decf873181819b12",
66);
	// Prueba 2B
	$listTags = $this->addCommentGame("yyyyy", 1);
	// Prueba 3B
	$listTags = $this->addCommentGame("63036cd97ef7a50387a0be48decf873181819b12",
-1);

	//PRUEBAS ADDGAME
	// Prueba 5
	$listTags = $this->addGame("63036cd97ef7a50387a0be48decf873181819b12", "aaaa", 68,
1, "14-06-2012 19:45:27");
	// Prueba 6
	$listTags = $this->addGame("yyyy", "aaaa", 66, 1, "14-06-2012 19:45:27");
	// Prueba 7
```

```
        $listTags = $this->addGame("63036cd97ef7a50387a0be48decf873181819b12", "a", 68, 1,
"14-06-2012 19:45:27");
        // Prueba 8
        $listTags = $this->addGame("63036cd97ef7a50387a0be48decf873181819b12", "aaaa", 66,
1, "14-06-2012 19:45:27");
        // Prueba 9
        $listTags = $this->addGame("63036cd97ef7a50387a0be48decf873181819b12", "aaaa", 68,
-1, "14-06-2012 19:45:27");
        // Prueba 10
        $listTags = $this->addGame("63036cd97ef7a50387a0be48decf873181819b12", "aaaa", 68,
1, "14062012194527");
        // Prueba 11
        $listTags = $this->addGame("63036cd97ef7a50387a0be48decf873181819b12", "aaaa", 68,
1, "50-13-2012 59:40:99");
        // Prueba 12
        $listTags = $this->addGame("63036cd97ef7a50387a0be48decf873181819b12", "aaaa", 68,
1, "");
        // Prueba 13
        $listTags = $this->addGame("63036cd97ef7a50387a0be48decf873181819b12", "", 68, 1,
"14-06-2012 19:45:27");

        //PRUEBAS DELCOMMENTGAME
        // Prueba 15
        $listTags = $this->delCommentGame("63036cd97ef7a50387a0be48decf873181819b12",
13);
        // Prueba 16
        $listTags = $this->delCommentGame("yyyyy", 13);
        // Prueba 17
        $listTags = $this->delCommentGame("63036cd97ef7a50387a0be48decf873181819b12",
-1);

        //PRUEBAS GETCOMMENTSGAME
        // Prueba 18
        $listTags = $this->getCommentsGame("63036cd97ef7a50387a0be48decf873181819b12",
66);
        // Prueba 19
        $listTags = $this->getCommentsGame("yyyyy", 1);
        // Prueba 20
        $listTags = $this->getCommentsGame("63036cd97ef7a50387a0be48decf873181819b12",
-1);

        //PRUEBAS GETTAGSGAME
        // Prueba 18-b
        $listTags = $this->getTagsGame("63036cd97ef7a50387a0be48decf873181819b12", 66);
        // Prueba 19-b
        $listTags = $this->getTagsGame("yyyyy", 1);
        // Prueba 20-b
        $listTags = $this->getTagsGame("63036cd97ef7a50387a0be48decf873181819b12", -1);

        //PRUEBAS GETCATEGORIES
        // Prueba 18-c
```

```
        $listTags = $this->getCategories("63036cd97ef7a50387a0be48decf873181819b12");
        // Prueba 19-c
        $listTags = $this->getCategories("yyyyy");

        //PRUEBAS GETGAMES
        // Prueba 21
        $fields = array();      $values = array();
        $listTags = $this->getListGames("63036cd97ef7a50387a0be48decf873181819b12", 0, 0,
$fields, $values);
        // Prueba 22
        $fields = array("idCategory");          $values = array(1);
        $listTags = $this->getListGames("63036cd97ef7a50387a0be48decf873181819b12", 1, 1,
$fields, $values);
        // Prueba 23
        $fields = array("name");        $values = array("a");
        $listTags = $this->getListGames("63036cd97ef7a50387a0be48decf873181819b12", 2, 2,
$fields, $values);
        // Prueba 24
        $fields = array("idGame");    $values = array(67);
        $listTags = $this->getListGames("63036cd97ef7a50387a0be48decf873181819b12", 3, 3,
$fields, $values);
        // Prueba 24 b
        $fields = array();      $values = array();
        $listTags = $this->getListGames("63036cd97ef7a50387a0be48decf873181819b12", 4, 0,
$fields, $values);
        // Prueba 24 c
        $fields = array();      $values = array();
        $listTags = $this->getListGames("63036cd97ef7a50387a0be48decf873181819b12", 5, 0,
$fields, $values);
        // Prueba 25
        $fields = array("idGame");    $values = array("a");
        $listTags = $this->getListGames("63036cd97ef7a50387a0be48decf873181819b12", 4, 3,
$fields, $values);
        // Prueba 26
        $fields = array("name");        $values = array(67);
        $listTags = $this->getListGames("63036cd97ef7a50387a0be48decf873181819b12", 5, 3,
$fields, $values);
        // Prueba 27
        $fields = array("name");        $values = array("a");
        $listTags = $this->getListGames("63036cd97ef7a50387a0be48decf873181819b12", -1, 2,
$fields, $values);
        // Prueba 28
        $fields = array("name");        $values = array("a");
        $listTags = $this->getListGames("63036cd97ef7a50387a0be48decf873181819b12", 2, -1,
$fields, $values);
        // Prueba 29
        $fields = array("name", "name");      $values = array("a", "b");
        $listTags = $this->getListGames("63036cd97ef7a50387a0be48decf873181819b12", 0, 2,
$fields, $values);
        // Prueba 30
        $fields = array("name", "idGame");  $values = array("a", "67");
```

```
        $listTags = $this->getListGames("63036cd97ef7a50387a0be48decf87318181 9b12", 0, 0,
$fields, $values);
        // Prueba 31
        $fields = array("name");        $values = array(67);
        $listTags = $this->getListGames("63036cd97ef7a50387a0be48decf87318181 9b12", 0, 2,
$fields, $values);
}
```

Anexo I. Manual de instalación

Paso a producción.

Las instrucciones siguientes tienen como objetivo servir de guía al/los administrador/res de kPax para instalar los cambios resultantes del proyecto en el entorno de producción. Junto con esta memoria se incluye el código del programa desarrollado por el proyectante. Los pasos que aquí se enumeran hacen referencia a los directorios y ficheros del código del proyectante y a los de la máquina de producción:

1. Hacer copia de seguridad de la carpeta elgg donde se incluye el código y módulos actuales de la parte cliente (elgg).
2. Localizar y proteger el script de creación de base de datos actual. Hacer igualmente una copia de seguridad de la base de datos actual.
3. Realice una copia de seguridad del código Java o localice la rama a la que se corresponde el código de producción en github.
4. Consulte y copie o anote la clave pública que se encuentra en */elgg/mods/kpax/lib/kpaxSrv.php* y las rutas a los servidores.
5. Copie el directorio elgg/mod/kpax/ del código implementado por el proyectante en la misma ruta del directorio de elgg actual.
6. Compruebe que las variables del fichero */elgg/mods/kpax/lib/kpaxSrv.php* apuntan correctamente a los servidores web de apache y de JBoss. Compruebe igualmente que la clave pública es la correcta.
7. Compruebe que las variables del fichero */elgg/mods/kpax/lib/kpaxOauth.php* apuntan correctamente a los servidores web de apache y de JBoss.
8. Ejecute el script pasoAProduccion.sql en la base de datos de kPax para añadir las nuevas tablas y campos diseñados en el proyecto. Como aclaración, existe un segundo script, kpac.sql (que no necesita ejecutar), que está pensado para sustituir el actual script kpac.sql presente en la carpeta doc de github y que genera toda la estructura de base de datos completa. El del código java también ha sido actualizado.
9. Inserte en la tabla Category las categorías que considere oportuno.
10. El código Java desarrollado por el proyectante está preparado para poder consultar juegos que tengan a NULL los valores de los campos idCategory y creationDate, añadidos en este proyecto a la tabla Game. No obstante, es conveniente editar sus valores en este punto, para evitar posibles errores a la hora de probar los cambios.
11. Copie o anote la clave privada contenida en el fichero */src/main/java/uoc/edu/svrKpax/util/ConstantsKPAX.java*
12. Sustituya el código Java por el nuevo.
13. *Compruebe que la clave privada del fichero ConstantsKPAX.java* sea correcta, sino, cámbiela por la anotada en el punto 11.
14. En el fichero pom.xml, compruebe que el elemento jbossdeployhome apunte a la carpeta de despliegue de JBoss.
15. Compile y despliegue el proyecto. Este paso es importante, porque el war contenido en el código entregado puede no ser correcto.
16. Pruebe la aplicación. Si encuentra algún fallo puede restaurar la base de datos, elgg y el código de Java, compilar de nuevo y desplegar de nuevo, corregir el error e intentar la instalación de nuevo.