Memoria prácticas
Programación Avanzada:
Apuestas

Adrián López Martínez
Brais Rey Souto
Enrique Paderne Granja

Índice

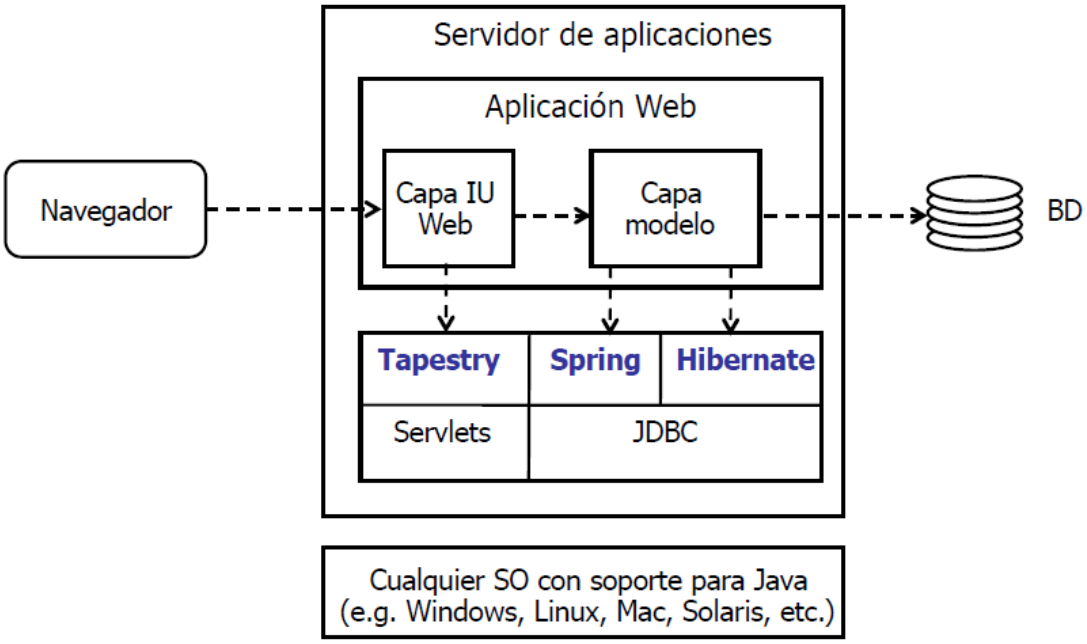
1. Introducción

Memoria sobre la práctica de la asignatura de Programación Avanzada. En ella se presentan tanto los diagramas utilizados como explicaciones del funcionamiento de un sitio Web de apuestas que realizamos. Esta permitirá registrar usuarios y estos podrán usar la página de apuestas para realizar apuestas sobre diferentes opciones de un evento.

Para diseñar e implementar el sitio Web usamos los frameworks Java POJO en el cual usamos Tapestry, Spring e Hibernate en vez de los propios de JAVA EE (Jpa, EJB y JSF). Esto nos permetirá una implementación más sencilla y rápida que sí usáramos EJB y JSF que son menos ágiles.

De esta manera el framework de Hibernate nos permitirá gestionar la persistencia de las entidades persitentes en la BD, Spring gestionará las transacciones y la inyección de dependencias entre beans y tapestry creará una interfaz para usar el servicio ofrecido por la capa modelo.

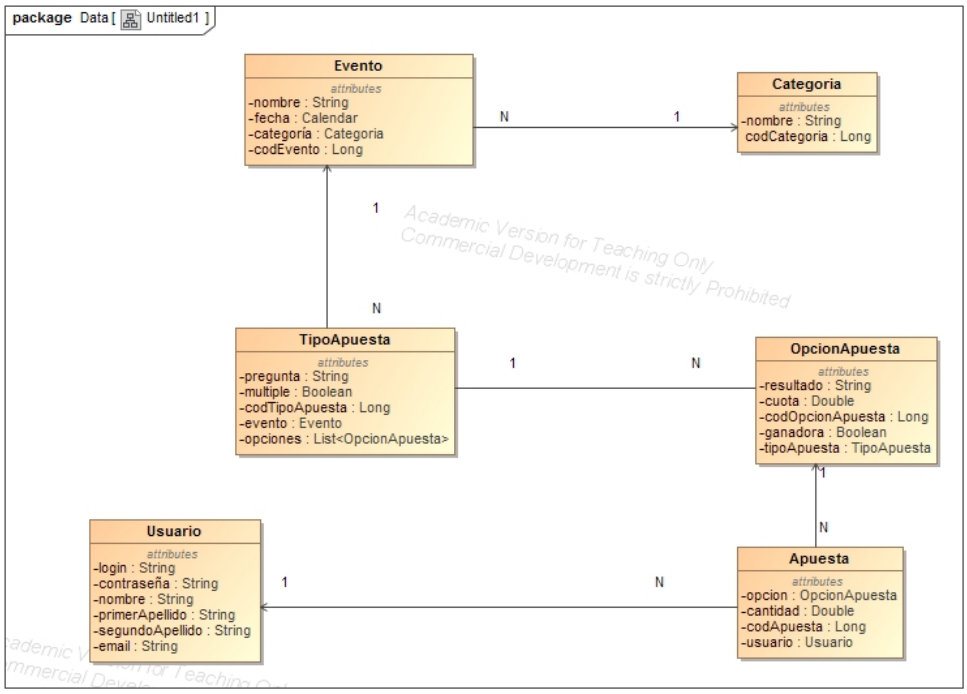
2. Arquitectura global



Para mostrar la estructura solo usamos este diagrama ya que el número de paquetes pertenecientes a cada capa es grande. En ella se ve como se comunican las diferentes capas y que frameworks usa cada una.
La aplicación está dividida en las siguientes capas: modelo, servicios e IU Web. La capa modelo y la de servicios la componen los ficheros dentro de **es.udc.pa.p007.apuestasapp.model.*** donde dividimos los pertenecientes a las entidades y a sus correspondientes daos, así como los servicios que van con terminación -service, de los cuales tenemos uno para la gestión de usuarios y otro para la gestión de eventos y de apuestas. La capa IU Web son los ficheros dentro de **es.udc.pa.p007.apuestasapp.web.*, los cuales contienen las páginas y los componentes utilizados para desarrollar la interfaz de la aplicación**.
En la capa modelo se encuentran las clases persistentes (*UserProfile, Apuesta, TipoApuesta, OpcionApuesta, Evento, Categoría*) con sus respectivas implementaciones de persistencia empleando DAOs gestionados con Hibernate.

3. Modelo

3.1. Clases persistentes

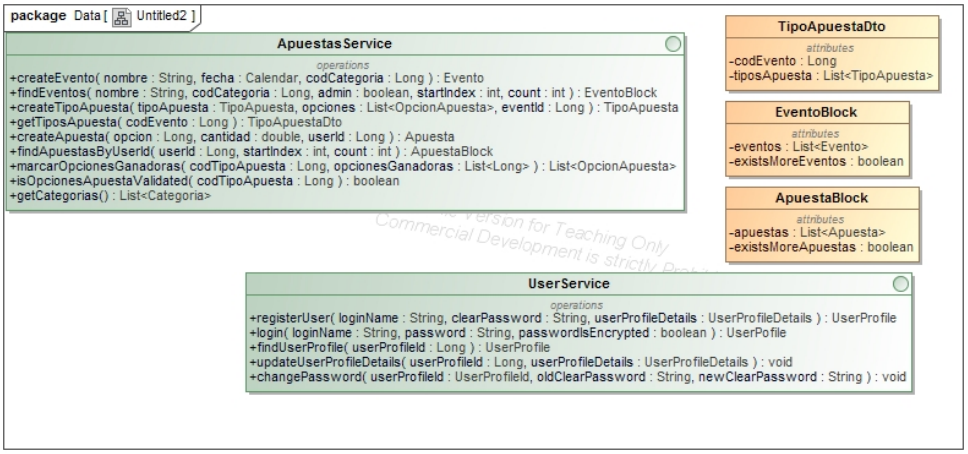


Las relaciones existentes entre las clases necesarias para gestionar la persistencia de los datos de la aplicación son UNO-MUCHOS/MUCHOS-UNO. Por otro lado, las relaciones se modelaron todas unidireccionales menos la existente entre TipoApuesta y OpcionApuesta que es bidireccional ya que se ha decidido dar la posibilidad de acceder directamente desde un Tipo de Apuesta a todas sus Opciones de Apuesta.

3.2. Interfaces de los servicios ofrecidos por el modelo

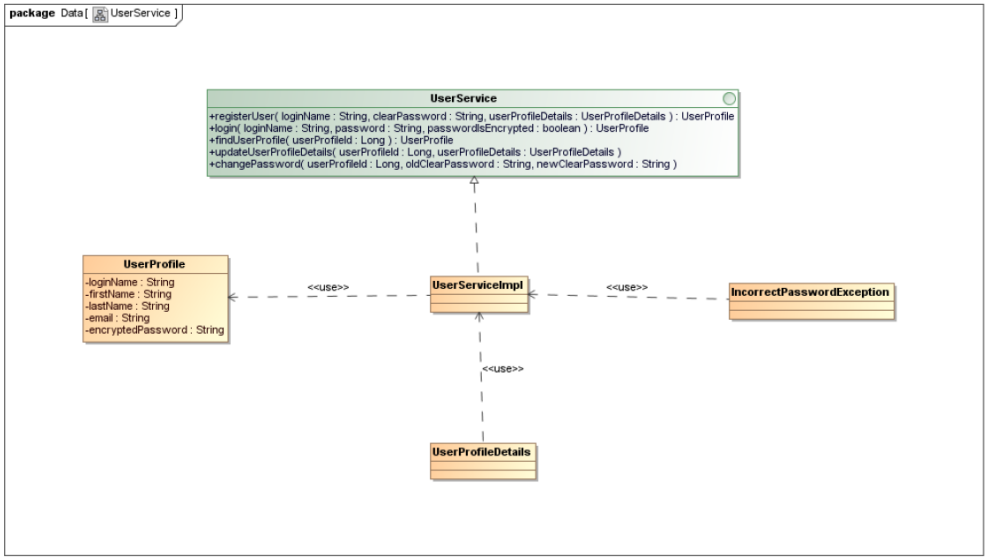
Para agrupar los diferentes servicios que ofrece la capa modelo se clasificaron en dos interfaces diferentes: una para representar las funcionalidades de la gestión de usuarios y otra para representar las funcionalidades propias de la aplicación.

Interfaz de las funcionalidades de la aplicación:



Esta muestra las funciones que usan los casos de usos de la aplicación. Aparte se han añadido algunas funciones como getCategorías que serán útiles para facilitar el desarrollo y simplifique la funcionalidad de la IU Web. Tambien se puede observar el DTO utilizado para obtener los tipos de apuesta de un evento. Y además tambien incluye los blocks usados para la paginación de eventos y de apuestas. Finalmente, tambien se incluye la interfaz de las funcionalidades del usuario, la cual se observa a continuación en más detalle.

Interfaz de las funcionalidades del usuario:



El usuario podrá realizar las funciones básicas iguales a las que contiene MiniPortal, ya que podrá registrarse, loguearse, actualizar su perfil, etc. y en definitiva, permite hacer una gestión de los usuarios y de las funcionalidades a las que tienen acceso cada uno de ellos.

3.3. Otros aspectos

3.3.1. Batchsize

Para gestionar el problema de las n+1 consultas (consume ram) se especificó que las entidades apuesta y evento, al hacer una recuperación se iniciarán 10 proxies (si existen) automáticamente. Como se paginan de diez en diez, se lanzarán dos consultas para recuperar toda la información, de la entidad relacionada y de 10 proxies en vez de 11. Con esto no saturamos la red con transacciones ni la memoria haciendo consultas cada vez que accedemos a un objeto de los paginados.

3.3.2. Inyección de dependencias

Para poder usar los elementos de los daos debemos acceder a ellos. Para eso Spring nos permite inyectar beans con la anotación @Autowired de clases que estén declaradas como beans.

3.3.3. Patrón Open Session On View

Para poder navegar entre objetos relacionados una vez obtenidos de la BD a través de un caso de uso, debemos permitir el uso de este patrón (descomentar la dependencia en WEB.XML). Si no se hace, si recuperamos las apuestas de un usuario, en el momento que accedamos a una de ellas (suponiendo que esta clase non está anotada como batchsize) no podremos obtener sus datos ya que ya se habrá terminado la transacción.

3.3.4. Programación orientada a aspectos AOP

Para gestionar las transacciones Spring permite hacer anotaciones en las clases que van a usar la BD y así no necesitamos el código de conexión a la BD ni el de inicio y fin de transacción. Para ello debemos anotar las clases que vayan a realizar transacciones como @Transactional.

4. Interfaz gráfica

Para la elaboración de la interfaz gráfica que se mostrará en el navegador al acceder a la aplicación usamos Tapestry. Tapestry compone la aplicación como un conjunto de páginas Web las cuales deben reaccionar a los eventos producidos. Cada página se compone internamente de dos archivos: un archivo Java que recibe los eventos de la página y una plantilla que genera los markup (lo que se mostrará en la página)

Existen casos de uso que solo pueden ser realizados por un tipo de usuario, en nuestro caso un usuario registrado como admin podrá realizar las acciones relacionadas con gestión, mientras el resto de usuarios serán los que puedan usar la aplicación para apostar o ver información. Las funciones disponibles en el sitio Web se estructuran en menús:

Autenticarse en un principio se mostrará este botón que nos permitirá identificarse en la aplicación (o registrarse) y al identificarse se cambiará por un menú con el nombre del usuario y las diferentes acciones que puede realizar sobre su perfil (Actualizar perfil, Cambiar contraseña o Salir)

Preferencias permite cambiar el idioma predeterminado que usa.

Buscar en este menú ya hay diferencias entre los diferentes usuarios, ya que un usuario normal podrá buscar eventos (futuros) y un historial con sus apuestas realizadas a lo largo del tiempo, el usuario registrado como admin podrá buscar todos los eventos y un usuario no registrado podrá buscar los eventos futuros.

Eventos este menú solo se le muestra al admin que podrá crear eventos.

5. Trabajos tutelados

5.1. Ajax

Ajax nos permite hacer las páginas interactivas: podemos realizar acciones sin que se tenga que recargar la página. Ajax permite solicitar información al servidor aunque la página haya sido cargada pudiendo realizar acciones como las implementadas en la práctica sin recargar la página o cargar otra nueva. Esta característica nos permite realizar:

Autocompletar texto Ajax permite autocompletar palabras. De esta manera podemos mostrar al usuario que esté realizando una búsqueda de eventos los eventos con los cuales coincidan los caracteres que ya haya introducido. Para eso la clase Java busca los nombres de los eventos que coinciden con los caracteres ya introducidos a tiempo real y se declara en la plantilla que el Field correspondiente a las palabras clave de la búsqueda se autocompleta. Como mucho se mostrarán 10 coincidencias.

Recargar zonas Ajax también permite que podamos recargar zonas de una página sin tener que recargar la página entera. Para ello debemos declarar la zona y cuando la clase Java reciba el evento que debe recargar la zona, indicarle a Ajax que recargue la zona. Esta opción la usamos en Crear Tipo Apuesta donde aparte de crear el tipo de apuesta, debemos crear la/s opción/opciones que deben acompañar el tipo de apuesta. De esta manera, declaramos la zona de creación de la opción de apuesta y cada vez que se crea una opción se recarga la zona habiendo guardado la opción creada en la sesión. Al finalizar de crear las opciones, podemos crear el tipo de apuesta con las opciones insertadas hasta el momento.

6. Problemas conocidos

Faltaría implementar una interfaz más amigable con imágenes y datos ya cargados cuando se acceda a la página para hacer publicidad.

La página de crear el tipo de apuesta en concreto, debería ser más intuitiva y fácil de usar para el usuario.