Memoria Pwin

Pablo Otero Aira
David Carballo López
Daniel Ferreiro Presedo

5 de junio de 2016

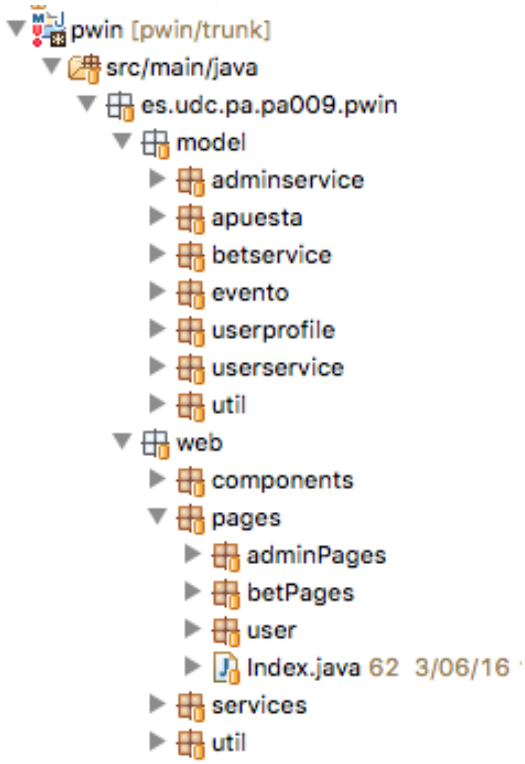
Índice

1. Arquitectura Global

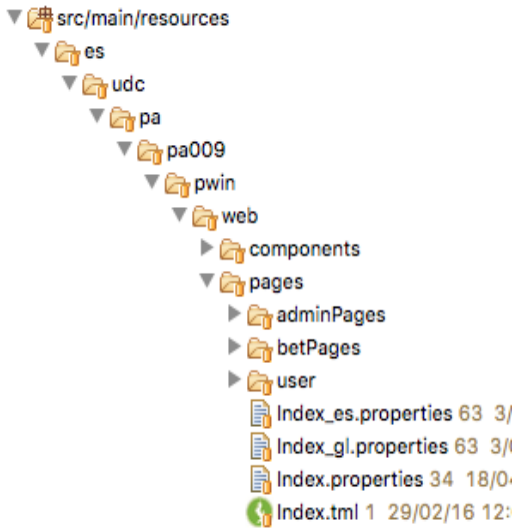
La estructura global del programa se divide principalmente en dos paquetes :

Model : este paquete incluye todo lo relacionado con la capa modelo (entidades persistentes y servicios). También están incluidos los DTO (Blocks) que se utilizan en determinadas funciones de la capa web.

Web: este paquete incluye lo relacionado con la capa web(clases Java de las páginas y los componentes, además de utilidades para el funcionamiento de la capa web, como permisos, autenticación, etc.), a excepción de los DTO mencionados anteriormente en la capa modelo.

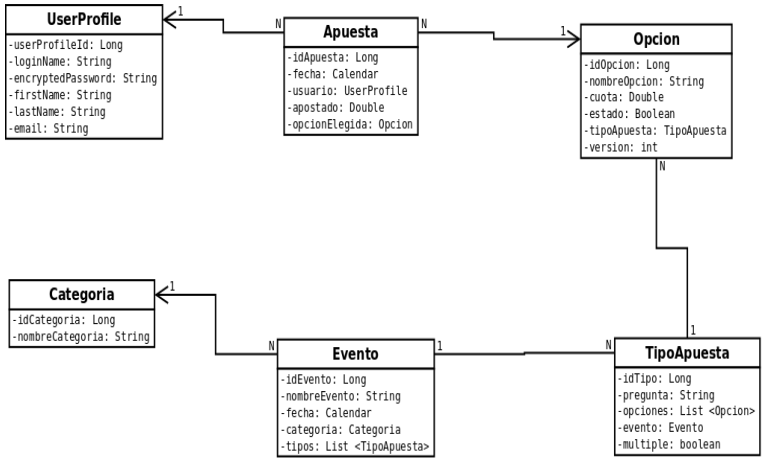


Dentro de la carpeta web, en el paquete de Resources, estarán incluidos los tml correspondientes a la clases página anteriormente mencionadas. Estarán divididos de forma análoga a sus hermanos del paquete Java.



2. Modelo

2.1. Clases Persistentes



2.2. Interfaces de los servicios

Los servicios de la aplicación fueron agrupados según el tipo de usuario (usuario normal o administrador) que utilizará las diferentes funcionalidades de la aplicación.

Los servicios ofrecidos por la aplicación son los siguientes:

BetService: contiene las funcionalidades a las que podrá acceder un usuario normal.

Utilizando esta interfaz, el usuario podrá realizar acciones como ver los eventos disponibles, apostar sobre ellos y comprobar sus apuestas realizadas.

AdminService: contiene las funcionalidades a las que podrá acceder un usuario con privilegios de administrador.

Con esta interfaz se permite que un administrador pueda crear eventos, nuevos tipos de apuesta con sus opciones correspondientes (sobre las que los usuarios apostarán) y marcar aquellas opciones ganadoras como tal.

A mayores, la aplicación cuenta con un servicio que permite el registro y la autenticación de usuarios, proporcionado por los profesores de la asignatura.

2.3. Otros aspectos

Page-by-page Iterator

Utilizamos el patrón Page-by-Page Iterator a la hora de mostrar los eventos y las apuestas realizadas por un usuario con la finalidad de que se reduzca los datos traídos a memoria, ofreciendo además al usuario una navegación más cómoda al mostrar los datos en páginas ordenadas.

FetchType

Para evitar que cuando se recupere un objeto de la Base de Datos también se recuperen todos aquellos que están relacionados con él seguiremos una política de navegación entre entidades LAZY. Esto implicará que cuando se recupere el objeto, en vez de recuperar las entidades enteras se creará un proxy (el cual almacenará el atributo que funciona de clave primaria y que permitirá recuperar el objeto entero en caso de necesidad)

BatchSize

Con el objetivo de evitar el problema de las 1+n consultas, hemos decidido optimizar la navegación entre las entidades a través del uso de la anotación de BatchSize. De esta manera conseguiremos que cuando se recupere un objeto de la Base de Datos, se aproveche la consulta para poder traer otros proxies del mismo objeto que todavía estén sin inicializar.

Hemos decidido aplicar esta optimización sobre las entidades de Evento, Categoría, Opción y TipoApuesta.

Optimistic Locking

Para permitir la modificación concurrente y evitar el problema de Second Lost Update hemos seguido la estrategia del Optimistic Locking, la cual nos permite evitar el problema sin aumentar el nivel de aislamiento de la Base de Datos (provocando menos bloqueos en la misma). Esto lo conseguiremos a través de la anotación de Version sobre un atributo de la entidad.

3. Interfaz Gráfica

En la capa vista, se utilizó la inyección de Spring para inyectar servicios de Tapestry y beans de Spring, concretamente los beans del servicio de la aplicación.

Para la capa vista se utilizó Tapestry, framework para el desarrollo de aplicaciones web, y Bootstrap (framework CSS/Javascript integrado con Tapestry) para posibilitar que la aplicación soporte Responsive Web Design, entre otras funciones.

A parte de los componentes proporcionados por Tapestry se creó el componente Layout, que genera el layout de las páginas y el contenido de las áreas genéricas, evitando definirlo en cada una de ellas. Con el fin de facilitar al usuario el uso de la aplicación hemos decidido insertar en el mismo el buscador de eventos, lo que le permite tenerlo siempre accesible. Además, en la parte superior derecha habrá un desplegable con su nombre que le dará acceso a todas las funcionalidades (que cambiarán según el tipo de usuario).

Open Session In View

Spring ofrece un filtro OpenSessionInViewFilter, que permite utilizar el patrón OSIV. Este filtro provocará que, para cada petición HTTP que intercepta, se abre una sesión de Hibernate, deja que fluya la petición, y cuando acaba, cierra la sesión. Este filtro fue habilitado en el *web.xml* manteniendo el orden de encadenamiento de los filtros definidos.

Sin embargo, con el fin de evitar que se realice una conexión a la Base de Datos por cada petición que llegue se modificará el archivo *spring-config.xml* para establecer una política LAZY. Esto provocará que sólo se abrirá la sesión de Hibernate cada vez que sea necesario, evitando tener así conexiones inútiles. Uso de GridDataSourc

Uso de Grids

Aprovechándonos del patrón Page-By-Page Iterator definido en la capa modelo hemos utilizado el componente *Grid* de Tapestry para mostrar los eventos y las apuestas realizadas del usuario, puesto que es sencillo de configurar a través de la implementación del GridDataSource (las cuales estarán ubicadas en el paquete util del paquete web de /src/main/java) y facilita la paginación.

4. Trabajos Tutelados

4.1. AJAX

Grids

Debido a que aplicamos el patrón Page by Page Iterator en ellos, el usuario no ve todos los eventos que existen en una sola página. Para evitar que esta se recargue mientras cambia entre ellas introducimos el atributo inPlace en los grids de cada .tml. Esto hace que unicamente se pidan los datos del grid en vez de recargar la página entera.

Creación de tipos de apuesta y opciones

Con el fin de reducir el número de peticiones y de hacer más sencilla la creación de los tipos de apuesta hemos declarado en la página 2 zonas que irán apareciendo a medida el usuario vaya realizando los pasos.

En primer lugar haberá un formulario (que permanecerá constante) para crear el tipo de apuesta, el cual provocará que se active la primera zona. En esta habrá un segundo formulario para crear las opciones asociadas a ese tipo de apuesta, lo que causará que cada vez que se añada una opción se refresque la segunda zona (que contendrá una tabla con las opciones insertadas hasta el momento, dando la opción de quitarlas)

Autocompletado de búsquedas

Para agilizar las búsquedas hemos implementado en el buscador de eventos una función de autocompletado. Para ello añadimos en el TextField encargado de recoger el nombre del evento el atributo *t:mixins="autocomplete"* e implementamos el evento onProvideCompletions, el cual devolverá las coincidencias con lo insertado por el usuario.

4.2. Selenium

Para implementar las pruebas automatizadas de la interfaz web hemos utilizado el IDE ofrecido por Selenium. Introduciendo en el *web.xml* las dependencias necesarias y aprovechando el plugin disponible para Mozzila hemos realizado los pasos que queremos que se realicen en el test. A partir de estos, el plugin generará los archivos correspondientes, los cuales tendremos que introducir en la carpeta test del paquete web del proyecto.

Para facilitar la ejecución de los test de la interfaz hemos implementado un perfil (webtest) para llevar a cabo los test mencionados. Para ello utilizamos el plugin de maven-surefire, que nos permitirá excluir los test que deseemos para cada uno de los perfiles. Además indicaremos que los test de la interfaz se realizan en una Base de Datos distinta (pojowebtest). Si queremos ejecutarlo deberemos realizar el siguiente comando: **mvn -Pwebtest test**

5. Problemas Conocidos

Almacenamiento de datos en la sesión

Para poder realizar ciertas funciones ofrecidas al usuario (como marcar las opciones ganadoras o crear las opciones del tipo de apuesta) necesitamos guardar datos en la sesión. Sin embargo, el problema surge cuando el usuario no acaba de realizar las acciones, puesto que los datos seguirán guardados en la sesión y podrán interferir con otra operación. Este problema se nos presenta debido a que no sabemos controlar cuándo el usuario le da a la opción de Atrás del navegador o cuando se está actualizando el contenido de la página.

Barra de búsqueda

A la hora de implementar la función de autocompletado de la búsqueda de eventos se nos ha modificado el header de la aplicación y se sobrepone al contenido de la página. Esto lo hemos intentando solucionar ampliando la zona definida para el header, pero también se podría arreglar modificando el CSS del mismo. Debido a que esto escapaba al ámbito de la práctica, hemos optado por la primera opción.