Memoria bet-fic

Felipe Bello Santos
Raquel López Cacho
Rubén Vigo Lamas

Universidade da Coruña

Programación Avanzada – Curso 2015/2016

Índice

1. Arquitectura Global

La aplicación se estructura en tres capas: la capa modelo o capa de acceso a datos, la capa servicios y la capa de interfaz del usuario.

La estructura de directorios del proyecto es la siguiente:

- **/bet-fic**: raíz del proyecto
- **/bet-fic/src/sql**: Contiene la implementación de la base de datos sql y la inserción de datos para las pruebas.
- **/bet-fic/src/test**: Contiene los test realizados para la capa modelo.
- **/bet-fic/src/main/java**: Contiene todos los subpaquetes con fichero Java de la capa modelo y la capa web.
- **/bet-fic/src/main/resources**: Contiene los ficheros de configuración de spring e hibernate. También contiene las páginas .tml que implementan la funcionalidad web del sistema.
- **/bet-fic/src/main/jetty**: Contiene la configuración de jetty.
- **/bet-fic/src/main/webapp**: Contiene la configuración para tapestry de la capa web en el fichero web.xml

Los paquetes más importantes y representativos de la implementación de la práctica son:

- **/bet-fic/src/test**: contiene los paquetes con las clases de pruebas.
- **/bet-fic/src/main/java**: paquetes que contienen la implementación del modelo y de la capa web en Java.
  - **Capa modelo:**
    es.udc.pa.pa002.practicapa.model.apuestarealizada: Clase persistente y DAO de ApuestaRealizada.
    es.udc.pa.pa002.practicapa.model.categoria: Clase persistente y DAO de Categoria.
    es.udc.pa.pa002.practicapa.model.evento: Clase persistente y DAO de Evento.
    es.udc.pa.pa002.practicapa.model.opcionapuesta: Clase persistente y DAO de OpcionApuesta.
    es.udc.pa.pa002.practicapa.model.tipoapuesta: Clase persistente y DAO

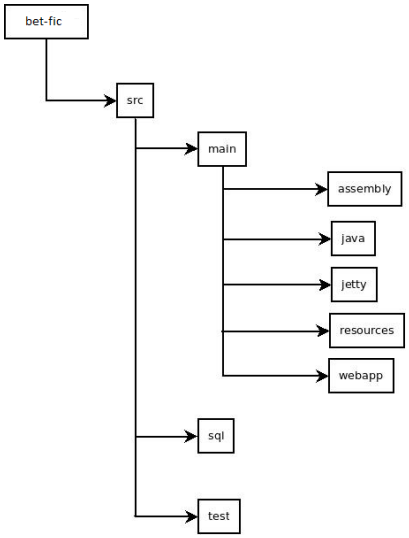

Figura 1: *Distribución por directorios del proyecto*

TipoApuesta.
es.udc.pa.pa002.practicapa.model.userprofile: Clase persistente y DAO de UserProfile.
es.udc.pa.pa002.practicapa.model.userservice: Clases correspondientes al servicio.

- **Capa web:**
es.udc.pa.pa002.practicapa.web.components: Paquete donde se ubican los componentes creados.
es.udc.pa.pa002.practicapa.web.pages: Paquete en el que están las páginas de la aplicación.
es.udc.pa.pa002.practicapa.web.pages.admin: Páginas en las que se ubican funcionalidades para el administrador.
es.udc.pa.pa002.practicapa.web.pages.useroperations: Páginas en las que se ubican funcionalidades para el usuario.
es.udc.pa.pa002.practicapa.web.pages.user: Páginas relacionadas con la gestión de los datos de gestión de los usuarios.

2. Modelo

2.1. Clases persistentes

En el siguiente diagrama podemos ver las clases persistentes y sus relaciones:

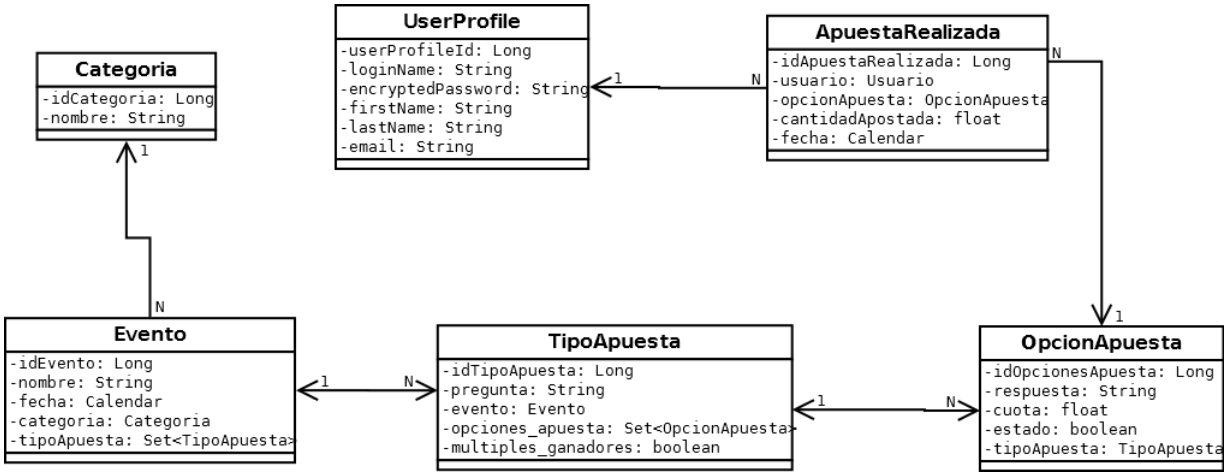

Figura 2: *Diagrama de clases persistentes del modelo*

2.2. Interfaces de los servicios ofrecidos por el modelo

Solo disponemos de una capa de servicios en la cual agrupamos todas los casos de uso, ya que se trata de una aplicación sencilla con un número reducido de funcionalidades. En las operaciones cuyo comportamiento varía según las invoque el administrador o un usuario normal, se emplea un booleano para especificar qué tipo de usuario la está invocando. De este modo podemos se evita realizar implementaciones distintas para ambos.

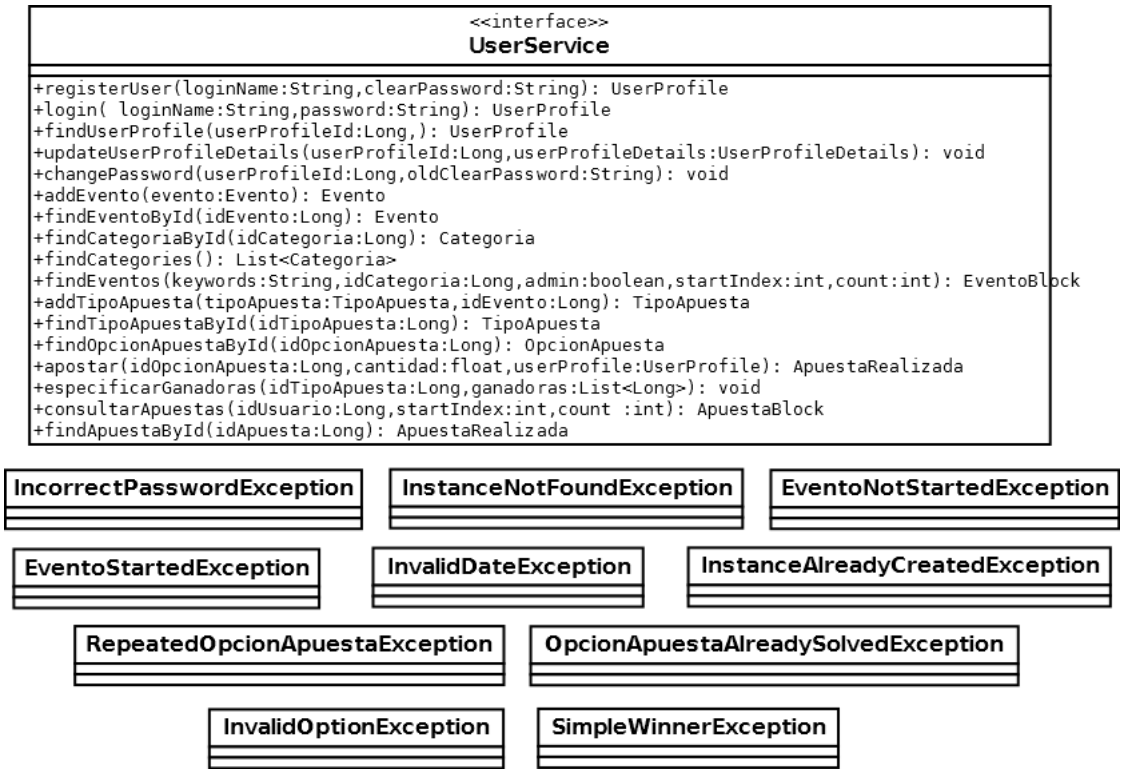

Figura 3: *Diagrama de la interfaz del servicio*

3. Interfaz Gráfica

El diseño de la interfaz gráfica de *bet-fic* está basado en *Tapestry* y *Bootstrap*, lo que nos permite minimizar el uso de CSS y html, además de mantener un diseño web Responsive.

Se distinguen **tres tipos de usuarios**: el usuario **sin autenticar**, el **autenticado** y el **administrador** de la web.

Los usuarios autenticados y sin autenticar tienen básicamente el mismo comportamiento, con la excepción de que en el momento de realizar una apuesta, si el usuario no está autenticado se le redirige a la página de login. Una vez que se autentique se le vuelve a redirigir a la página en la que se encontraba para que pueda realizar su apuesta. Ambos puesden buscar y visualizar eventos, filtrando por palabras clave o por categoría. Si no se establece ningún criterio de búsqueda se muestran todos los eventos. A los usuarios normales se les

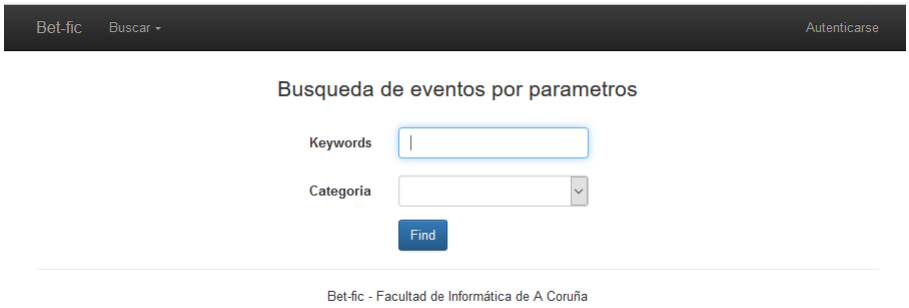

Figura 4: *Formulario de búsqueda de eventos (plantilla findEventos.tml)*

mostrará la lista de eventos coincidentes con la búsqueda que todavía no hayan comenzado. Al seleccionar un evento concreto, se mostrará su información, con

Figura 5: *Lista de eventos coincidentes con una búsqueda (plantilla foundEventos.tml)*

los tipos de apuesta y opciones asociadas. Los usuarios podrán seleccionar una

Figura 6: *Visualización de los detalles de un evento (plantilla eventoDetails.tml)*

opción para realizar una apuesta sobre ella. Se mostrará una página con un formulario para la realización de esta operación.

Figura 7: *Formulario para realizar una puesta (plantilla opcionApuestaDetails.tml)*

Los usuarios que estén autenticados pueden ver un histórico de sus apuestas con el resultado de cada una. Pueden estar pendientes (—), o resueltas. En el caso de que estén resueltas se visualizará claramente si la resolución fue positiva o negativa. El administrador podrá realizar búsquedas de eventos del

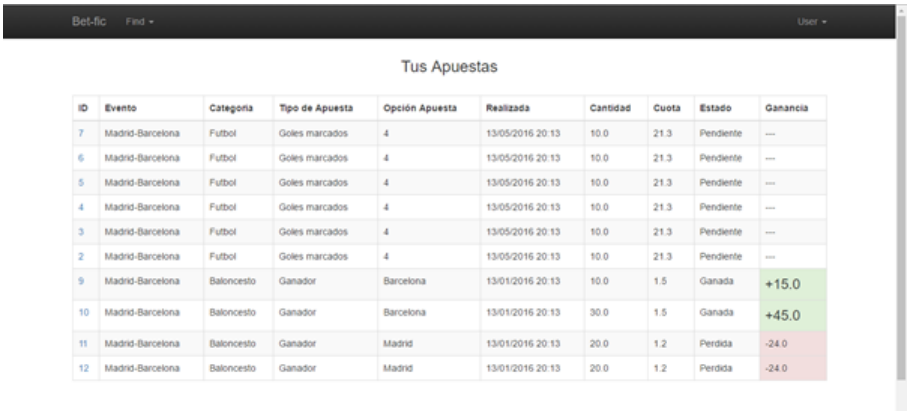

Figura 8: *Formulario para realizar una puesta (plantilla findApuestasUser.tml)*

mismo modo que los usuarios normales, con la diferencia de que podrá ver todos los eventos (independientemente de su fecha de inicio). Por otra parte, será el encargado crear los nuevos eventos, sus tipos de apuestas y sus opciones asociadas. Distinguimos dos tipos de apuesta, apuesta simple (en la que solo

Figura 9: *Formulario para añadir opciones de apuesta (plantilla anadirOpcionApuesta.tml)*

habrá una opción ganadora) y apuesta múltiple (en la que podrá haber varias opciones ganadoras). A la hora de establecer las opciones ganadoras (función llevada a cabo por el administrador) la interfaz se comportará de forma distinta según sea múltiple o simple ganadora.

En el caso de que sólo se permita una opción ganadora usamos el componente

Figura 10: *Página para mostrar la información de los tipos de apuesta creados (plantilla TipoApuestaCreado.tml)*

de Tapestry *radio button*, que sólo permite seleccionar una opción a la vez.

Figura 11: *Establecer opciones ganadoras de un tipo de apuesta simple (plantilla EspecificarGanadoras.tml)*

Si se permiten múltiples opciones ganadoras, se usa el componente *checklist*, que permite seleccionar varios elementos a la vez.

Figura 12: *Establecer opciones ganadoras de un tipo de apuesta múltiple (plantilla EspecificarGanadoras.tml)*

Una vez especificadas las opciones ganadoras, éstas se resaltan entre las demás.

Figura 13: *Opciones ganadoras resaltadas (plantilla TipoApuestaDetails.tml)*

Para tener retroalimentación con el usuario mostramos mensajes de errores en rojo y notificaciones de satisfacción en verde.

Figura 14: *Mensaje de evento creado correctamente (plantilla EventoCreated.tml)*

Figura 15: *Muestra de errores (plantilla AnadirEvento.tml)*

4. Trabajos tutelados

4.1. Ajax

Hemos aplicado el enfoque AJAX en los siguientes casos :

- **Búsqueda de eventos:** hemos añadido una interacción AJAX que autocompleta las keywords introducidas por el usuario según los eventos disponibles en ese momento. Para implementarla hemos empleado el com-

ponente ***mixins*** de Tapestry (concretamente la función ***autocomplete***). Este componente lanza un evento *providecompletions*, que capturamos en la clase java asociada a la plantilla, con el método *onProvideCompletionsFromKeywords*. En este método se hace una llamada al servicio que proporciona los valores de autocompletado mostrados.

- **Paginación en la visualización de eventos**: para mostrar la lista de eventos resultado de una búsqueda, hemos utilizado el componente ***grid*** de Tapestry, que permite realizar la paginación de los resultados de búsqueda de forma automática (mediante el uso del parámetro rowsPerPage). Este componente ya incorpora soporte para interacciones AJAX. Puede activarse estableciendo el parámetro ***inPlace*** a true. De esta forma la paginación se realiza con AJAX, recargando la zona del grid en lugar de toda la página.

- **Añadir opciones a un tipo de apuesta:** Cuando el administrador crea un tipo de apuesta (con sus opciones asociadas), en primer lugar se presenta una página con un formulario para introducir los datos del tipo de apuesta.A continuación, se muestra otra página para ir introduciendo las respectivas opciones. Dicha página contiene un formulario para introducir los datos de la opción y una tabla que se va completando con cada valor introducido. Para optimizar este caso de uso, y que no sea necesario

recargar la página cada vez que el administrador introduce una opción, hemos empleado el componente de Tapestry ***Zone***.

4.2. Pruebas funcionales contra la interfaz Web

Utilizamos el componente WebDriver de la herramienta Selenium para implementar las secuencias de navegación y el framework JUnit para realizar los casos de prueba. Para poder usar la herramienta añadimos la dependencia correspondiente a Selenium en el **pom.xml** de nuestro proyecto.

Implementamos dos casos de prueba:

- **Autenticación de un usuario:** consiste en acceder a la página de login, autenticarse y comprobar que la aplicación muestra el nombre del usuario.
- **Realización de una apuesta:** se realiza una búsqueda de eventos en función de unas palabras clave introducidas, accedemos a un tipo de apuesta de un evento y apostamos por una opción de apuesta. La aplicación nos redirige a la página de autenticación, nos autenticamos correctamente y finalmente realizamos la apuesta. Tras la confirmación de la apuesta ejecutamos el caso de uso de buscar las apuestas de un usuario, para comprobar que los datos son correctos. Comparamos los datos de la primera apuesta mostrada, que se corresponde con la última realizada, con los datos correctos.

Para ejecutar estas pruebas de forma independiente de las pruebas de integración, creamos un nuevo perfil maven, el **perfil webtest**. Para establecer la configuración de este perfil creamos un tag *profile* dentro de *profiles* del **pom.xml** de nuestro proyecto.

El objetivo es que cuando se ejecuten tests con el perfil por defecto sólo se ejecuten los de la capa modelo y con webtest sólo los web. Para ello, configuramos los perfiles para que introduzcan en una variable los tests que queremos que ejecuten. El valor de dicha variable lo introducimos en el tag *include* del plugin de maven.

En la configuración del perfil webtest, a diferencia de el de por defecto, el valor del tag *datasource.url* será el de la base de datos sobre la que ejecutamos las pruebas, **pojowebtest**. De esta manera, el servidor de aplicaciones se arranca sobre pojowebtest cuando usamos este perfil, y así dispone de los datos necesarios para las pruebas de forma independiente.

El tag *testdatasource.url* lo mantenemos con *pojotest*, igual que el perfil inicial, aunque no usamos la base de datos.

Para ejecutar las pruebas contra la interfaz ejecutamos **mvn -Pwebtest sql:execute**. De esta manera introducimos los datos de los casos de prueba en la base de datos. Posteriormente iniciamos el correspondiente servidor de aplicaciones con **-Pwebtest**, y para la ejecución de la clase de prueba **mvn -Pwebtest test**.

5. Problemas conocidos

La internacionalización solo está en la página de login.

Los usuarios deben permitir la ejecución de código jacascript en su navegador. Si no lo hacen, la aplicación no podrá exponer todas sus funcionalidades.