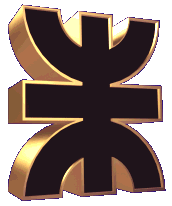
|  |
| --- |
| **Nombre y Apellido** |
| Cappellini Darío |
| Carella Germán |
| Centurion Damian |
| Gonzalez Santiago |

|  |  |
| --- | --- |
| **Fecha De Presentación:** | 07/10/2014 |
| **Fecha de Devolución:** |  |
| **Calificación** |  |
| **Firma Profesor** |  |



**GRUPO N° 7**

TP Anual: 7ma Entrega

***Diseño de Sistemas 2014***

***Miércoles mañana***

**Correcciones sobre la entrega anterior**

Se creo una Home de Comunidad para poder manejarla dentro de la aplicación. Ésta misma engloba las otras dos Home (Jugadores y Partido), que se eliminan.

Se crea con el objetivo de tener un único ambiente para todos los jugadores y los partidos que se van jugando, aprovechando el concepto propio de la comunidad, permitiendonos utilizar adecuadamente todos los criterios de búsqueda y de generación de equipos.

Esta Home permite:

* Conocer todos los jugadores dentro de la comunidad.
* Crear partidos para inscribir estos jugadores. Esto permite saber cuál fue el último partido.
* Setear a cada jugador un valor de “Handicap” (para el criterio de selección por handicap).
* Setear a cada jugador un apodo (para la búsqueda por apodo).
* Crear amistades entre los distintos jugadores dentro de la comunidad (para el detalle de cada jugador -> lista de amigos).
* Aplicar infracciones a cada jugador (para el detalle de cada jugador -> infracciones).
* Calificar entre jugadores (para generar los promedios generales y del último partido).

Todas estas consideraciones que nos permiten la nueva Home, no se estaban teniendo en la entrega anterior, pero uniendo todo en una sola, pudimos reutilizar lo generado en entregas anteriores.

**Layout**

Al utilizar el framework Wicket, el diseño gráfico y posicionamiento de los componentes se resuelve simplemente mediante el .html. Este framework permite armar todo el layout por una parte, sin preocuparse por la funcionalidad de la aplicación, y luego agregar el comportamiento colocando los Wicket ID a los tags que correspondan.

**Separación entre modelo y vista**

Se logra mediante la aplicación del patrón MVC o MMVC. Dejamos la lógica del dominio intacta y creamos algunos objetos de dominio adicionales (los Application Model) donde resultó necesario. La vista simplemente envía mensajes al dominio (mediante el funcionamiento interno de Wicket, que también se encarga de conservar el estado) pero no se guarda datos de dominio.

**Manejo del estado conversacional de los datos que se guardan en la vista**

Los datos que se “guardan en la vista” en realidad no están en la vista: lo que sucede es que cuando se ingresa un cierto dato se bindea con un atributo de un objeto de dominio. Wicket tiene mecanismos transparentes para nosotros que permiten guardar todo el estado de los objetos de dominio que se creen y utilicen durante una sesión.

**Tipo de binding utilizado**

Se utiliza binding bidireccional ya que permite un flujo de datos desde la vista hacia el modelo y viceversa. Aunque no es posible un binding bidireccional en tiempo real por el hecho de que HTTP es un protocolo unidireccional, Wicket aprovecha las ocasiones en las que se refresca una página o se navega entre páginas (es decir, cuando corresponde hacer un pedido) para enviar al servidor web las actualizaciones. Nosotros podemos considerarlo como binding bidireccional aunque esto sea discutible porque Wicket nos oculta muy bien los detalles ásperos de la adecuación a HTTP.

**Manejo de eventos**

Al ser una aplicación bastante simple, los únicos eventos que manejamos son clicks, ya sea sobre botones o sobre links. Los objetos que usamos para implementar estos componentes gráficos nos ofrecen redefinir su “onClick”. Al ser éste un bloque de código que podemos definir, podemos enviar cualquier conjunto de mensajes necesarios para manejar el evento al hacer un click en estos componentes.

**Navegación**

Implementamos la navegación entre páginas mediante el mecanismo de “responsePage” que ofrece Wicket, el cual nos permite navegar a una página determinada. Podemos setear como “responsePage” a una nueva página que estamos creando, o a una página que ya conocemos. Por ejemplo, todas nuestras páginas que implementan un botón volver reciben por parámetro al ser creadas una “parentPage” que no es más que la página que la llamó. Entonces, al tocar el botón volver, con la sentencia *responsePage = parentPage* Wicket permite regresar a la página anterior que, como es un objeto que se mantiene vivo en el ambiente, conserva el estado que tenía la última vez que la visitamos.

Además de facilitar el regreso y la creación de nuevas páginas, el hecho de que las páginas sean objetos nos permiten establecer un flujo de información. Al navegar “hacia adelante” (es decir en todos los casos que no se trata de un botón volver) estamos ejecutando una línea que crea un nuevo objeto página. Al crearlo, le podemos pasar toda la información relevante en el constructor.

Como ejemplo de esto está la página que muestra los datos del jugador. Supongamos que una página tiene una referencia al jugador Jorge, y quiere navegar hacia la página que muestre sus datos. Como la página que muestra los datos es obviamente una sola, necesita recibir la información de a qué jugador tiene que mostrar. En este caso, lo que necesitamos ejecutar es simplemente *responsePage = new InfoJugadorPage(jorge, this)* donde *jorge* es la información que necesitábamos comunicar y *this* es la parentPage necesaria para saber a dónde hay que volver.