# Práctica Web .NET – Sitio web de comentarios de apuestas deportivas Integración de Sistemas 2009-2010

Barros Barros, Ismael Fernández Núñez, Daniel Iglesias Fraga, David

2 de julio de 2010

## ${\bf \acute{I}ndice}$

| 1.         | Arquitectura global                        | 5          |
|------------|--|------------|
| 2.         | Modelo  2.1. Clases persistentes           | 8          |
| 3.         | Interfaz gráfica                           | 13         |
| 4.         | Partes adicionales                         | <b>1</b> 4 |
| <b>5</b> . | Compilación e instalación de la aplicación | 14         |
| 6.         | Problemas conocidos                        | 15         |

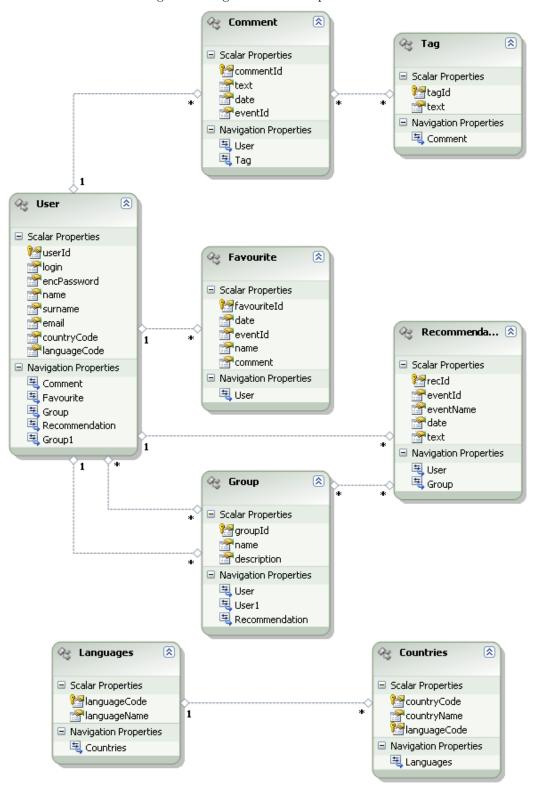
#### 1. Arquitectura global

Figura 1: Diagrama de arquitectura CommentDao GroupDao pruebas para cada fachada Test Una clase de TagDao DAOs para las entidades modeladas con Entity Data Model FavouriteDao Librerias incluidas, en este caso solamente ModelUtil.dll Гiв Es.Udc.DotNet.Betbook UserDao RecommendationDao sesiones de usuario y la internacionalización Utilidades para implementar las **BetOMaticService** Contiene las páginas de error HTTP Servicio usado para acceder al servicio REST de la parte de Java Web Errors CommentService Pages XSLT Contiene todas las páginas de la Web GroupService Archivos para hacer las transformaciones XSL Servicios ofrecidos a la capa web. Cada uno de ellos tendrá su clase de EventService pruebas Exceptions UserService

#### 2. Modelo

#### 2.1. Clases persistentes

Figura 2: Diagrama de clases persistentes



#### 2.2. Interfaces de los servicios ofrecidos por el modelo

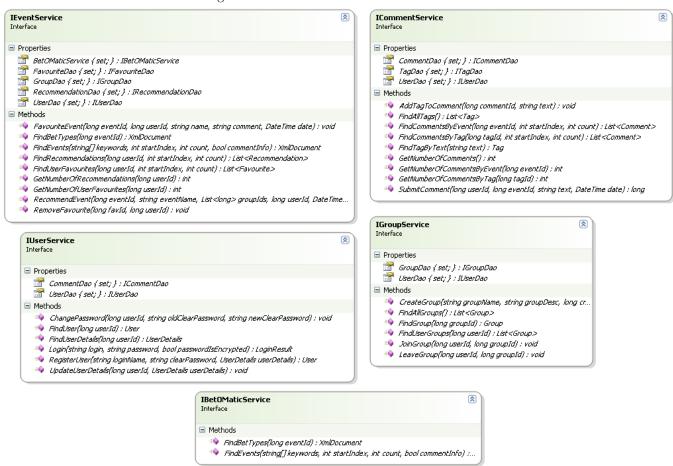


Figura 3: Interfaces de los servicios

Se ha clasificado la funcionalidad del sistema en tres servicios, siguiendo un criterio semántico para su segmentación:

Servicio de usuario: Contiene la funcionalidad referida a los perfiles de los usuarios del sistema. Permite, por ejemplo, su registro, autenticatión y modificación

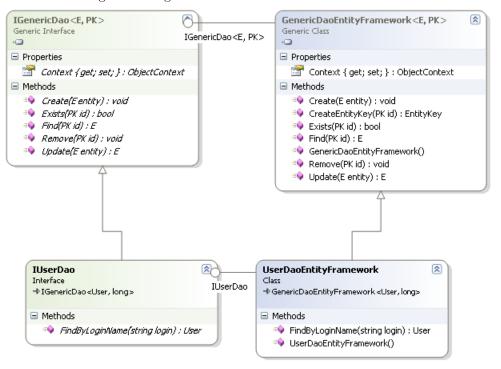
Servicio de grupos: Contiene la funcionalidad de gestión de grupos de usuarios. Permite su creación, y que un usuario dado se una o deje un grupo.

Servicio de comentarios: Contiene la funcionalidad relacionada con la gestión de los comentarios y sus etiquetas. Los comentarios pueden ser creados, etiquetados, y buscados por evento o por etiqueta.

Servicio de eventos: Contiene la funcionalidad relacionada con eventos, las recomendaciones de los mismos y los favoritos.

#### 2.3. Diseño de un DAO

Figura 4: Diagrama de clases del DAO de la clase User



#### 2.4. Diseño de un servicio del modelo

IUserDao Interface → IGenericDao <User, long> IUserService Interface UserDaoEntityFramework ♠ Class

→ GenericDaoEntityFramework < User, long> IUserDao IUserDao Properties CommentDao { set; } : ICommentDao
 UserDao { set; } : IVserDao
 Methods ■ Methods 🗣 FindβyLoginName(string login) : User 🗣 FindByLoginName(string login) : User UserDaoEntityFramework() ■ Methods ChangePassword(long user[d, string oldClearPassword, string newClearPassword) ; void FindUser(long user[d) : User
 FindUserDetails(long user[d) : UserDetails × LoginResult Class | Login(string login, string password, bool passwordsEncrypted) ; LoginResult
| RegisterUser(string loginName, string clearPassword, UserDetails userDetails) ; User
| UpdateUserDetails(long userId, UserDetails userDetails) ; void
| ■ Properties Properties

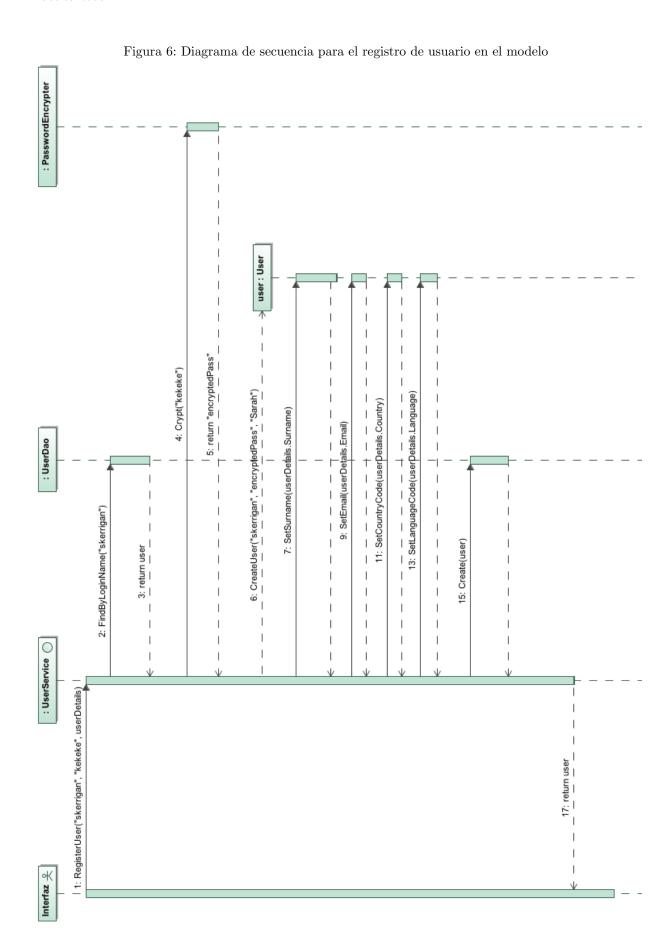
Country { get; set; } : string

EncryptedPassword { get; set; } : string

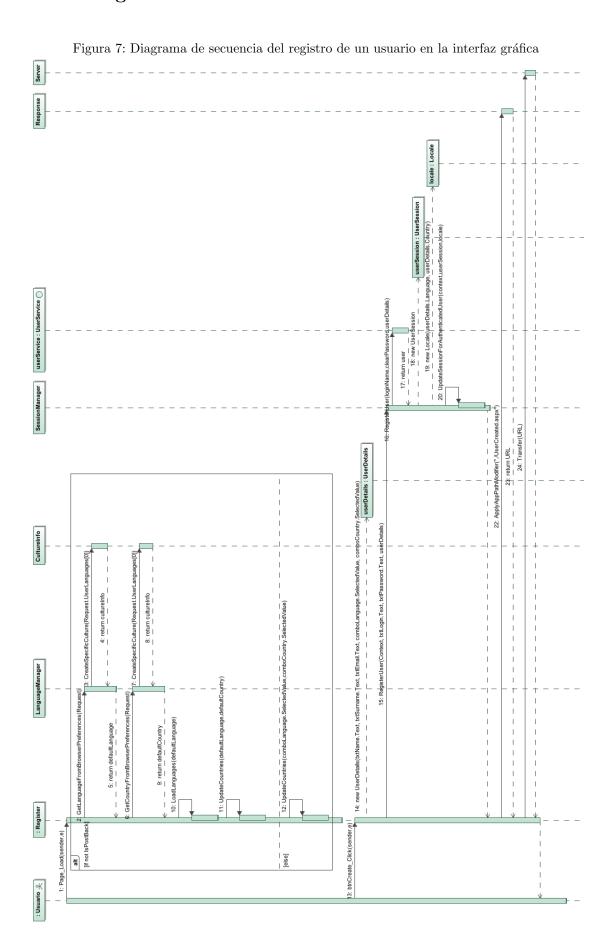
Name { get; set; } : string

VerId { get; set; } : long UserService Class ■ Methods CommentDao { get; set; } : ICommentDao Equals(object obj) : bool GetHashCode(): int ■ Methods LoginResult(long userId, string name, string encryptedPassword, string language, string country) ChangePassword(long userProfileId, string oldClearPassword, string newClearPassword): void ToString(): string FindUser(long userId) : User FindUser(Details(long userProfileId): UserDetails
 Login(string login, string password, bool passwordIsEncrypted): LoginResult RegisterUser(string login, string clearPassword, UserDetails userDetails): User UserDetails Class <u>^</u> UpdateUserDetails(long userId, UserDetails newUser) : void ■ Properties Country { get; set; } : string
Country { get; set; } : string ICommentDao ICommentDao Interface → IGenericDao <Comment, long> **☆** ■ Methods ■ Methods Equals(object obj) : bool FindByEventIdflong eventId, int startIndex, int count): List<Comment> GetHashCode(): int FindByTagId(long tagId, int startIndex, int count): List<Comment>
 GetNumberOfComments(): int ToString(): string UserDetails(string firstName, string lastName, string email, string language, string co. GetNumberOfCommentsByEvent(long eventId) ; int
 GetNumberOfCommentsByTag(long tagId) ; int CommentDaoEntityFramework Class
→ GenericDaoEntityFramework < Comment, long> FindByEventId(long eventId, int startIndex, int count): List<Comment>
 FindByTagId(long tagId, int startIndex, int count): List<Comment> GetNumberOfComments(): int GetNumberOfCommentsByEvent(long eventId): int
 GetNumberOfCommentsByTag(long tagId): int

Figura 5: Diagrama de clases del servicio UserService



## 3. Interfaz gráfica



#### 4. Partes adicionales

#### Parsing de XML:

Se le permite al usuario visualizar información sobre los comentarios de cada evento en la lista de resultados del caso de uso de búsqueda de eventos.

Este caso de uso está implementado en la capa modelo obteniendo un *XmlDocument* a partir del servicio XML de la parte de Java de la asignatura, y devolviéndoselo intacto a la capa web para que le aplique una transformación XSLT.

Para implementar la parte opcional, la capa web le indica al modelo mediante un parámetro que el usuario requiere información de comentarios, y éste, una vez ha obtenido el *XmlDocument*, itera sobre cada uno de los eventos devueltos (//event), buscando el número de comentarios existentes para cada evento e inyectándolo en el mismo *XmlDocument* en un nuevo elemento //event/numberOfComments.

Una implementación alternativa habría sido crear una lista de objetos que fusionen la información de los eventos devueltos con información sobre comentarios, pero la solución propuesta se ha considerado más limpia y fácil de implementar, una vez codificada la parte obligatoria de la práctica.

#### Etiquetado de comentarios:

- Al añadir un comentario, el usuario dispone de una caja de texto adicional donde puede insertar etiquetas separadas por espacios o por comas, siguiendo un diseño similar al que siguen sitios como <a href="http://www.flickr.com">http://www.flickr.com</a>. Adicionalmente, se le expone al usuario una lista con los etiquetas existentes. Cada etiqueta es un link que ejecutará una pequeña rutina escrita en Javascript, que simplemente añadirá la etiqueta a la caja de texto de etiquetas.
- En todo momento, se expone bajo el menú principal de la aplicación una nube de etiquetas, con las etiquetas existentes en base de datos dispuestas en orden arbitrario pero con un tamaño de letra directamente proporcional al número de comentarios que fueron etiquetados con esa etiqueta.
- Al visualizar la lista de comentarios para un evento, se muestra para cada comentario la lista de etiquetas asociadas al mismo.

En todos los casos, la lista de etiquetas se ha implementado como un simple ListView de ASP.NET, que toma como DataSource una lista de objetos Tag. En los dos últimos casos, cada etiqueta es un link que le permite al usuario ver la lista de comentarios etiquetados con la misma.

## 5. Compilación e instalación de la aplicación

En un entorno con el software necesario instalado y configurado correctamente se han de seguir los siguiente pasos para compilar e instalar la aplicación:

- 1. Ejecutar el servidor de la parte Java:
  - $a) \ \$ \ {\tt mysqld} \ --{\tt defaults-file=\$HOME/.my.cnf}$
  - b) \$ mvn jetty:run
- 2. Ejecutar la aplicación:
  - a) Botón derecho sobre el proyecto "Web" > Set as StartUp Project
  - b) Botón derecho sobre "Pages/MainPage.aspx" > Set as Start Page
  - c) Ejecutar el proyecto "Web" (Ctrl + F5)

### 6. Problemas conocidos

■ La implementación de la nube de etiquetas es ineficiente, porque en cada llamada a la página, se cuenta el número de comentarios asociado a cada uno de las etiquetas, con el fin de renderizar cada etiqueta con un tamaño de letra proporcional al número de comentarios. Este detalle se podría solucionar almacenando en cada etiqueta el número de comentarios asociados al mismo, y actualizando este contador mediante triggers.