Ampliación de Bases de Datos
Año 2015/2016
Grado en Ing. Informática
Fdi - UCM

Práctica 1

Para solteros exigentes

Fecha de entrega: Esta práctica tiene varias entregas. Consulta la última página del enunciado para comprobar la fecha de cada una. La entrega final tiene como fecha tope el 29 de abril de 2016.

Porcentaje en la calificación de prácticas: 60 % (1.80 puntos en la calificación final de la asignatura).

Calificación: Las dos primeras entregas se corresponden, respectivamente, con los apartados 1 y 2 del enunciado. No existe nota numérica asociada a las dos primeras entregas; serán calificadas como APTO o NO APTO. Una calificación de NO APTO podrá ser recuperada en la entrega final. La última entrega comprende los restantes apartados de la práctica, de los cuales son obligatorios el 3º y 4º. Con estos apartados se puede obtener una calificación de APTO con una nota máxima de 4 sobre 10 (0.72 puntos en la nota final de la asignatura). Para optar a una nota superior se deberán implementar las extensiones de la práctica. La calificación máxima varía en función de los apartados implementados:

- Apartados 1 a 5: Calificación máxima de 5/10 (0.90 puntos en la nota final de la asignatura).
- Apartados 1 a 6: Calificación máxima de 7/10 (1.26 puntos en la nota final).
- Apartados 1 a 7: Calificación máxima de 8/10 (1.44 puntos en la nota final).
- Todos los apartados: Calificación máxima de 10/10 (1.80 puntos en la nota final).

Se valorará la corrección del diseño de la base de datos y la correcta división de responsabilidades en la aplicación. Se valorarán negativamente las referencias directas al diseño de la BD relacional que no esté contenidas en las clases de la aplicación que se encargan del acceso a datos.

Se recuerda que la ausencia de entregas, o la entrega de una práctica total o parcialmente copiada conllevará la calificación de NO APTO no recuperable.

El objetivo de esta práctica es el diseño de una red social de citas. Los usuarios/as de esta red tendrán la oportunidad de conocer a otras personas cercanas, consultar sus respectivos perfiles, y contactar con ellas. La red social también dispondrá de un sistema de preguntas y respuestas. Un usuario puede contestar voluntariamente a algunas de las preguntas contenidas en la base de datos. Las respuestas proporcionadas permitirán posteriormente realizar un cálculo de compatibilidad entre dos usuarios.

La práctica tiene un objetivo doble:

1. Diseñar una base de datos (de complejidad media-alta) para modelar las entidades y relaciones que forman parte de la lógica de esta red social.
2. Diseñar e implementar una aplicación en Java que acceda a esta base de datos mediante JDBC. Para ello utilizaremos:

    a) Patrones de acceso a datos (Tema 2).

b) La herramienta de mapeo objeto-relacional *Hibernate* (Tema 3).

A continuación se describe el tipo de información que se debe almacenar para los principales elementos de la base de datos.

- En primer lugar tenemos, obviamente, a los **usuarios** de la red social. Cada usuario queda identificado por su dirección de correo electrónico, que no es visible para los demás usuarios de la red. El usuario podrá ingresar (*log in*) en la red mediante una dirección de correo electrónico y una contraseña. Otros datos obligatorios para el usuario son: su nombre (visible a los demás usuarios de la red), su género (masculino o femenino), y el de las personas con las que desea contactar (hombres, mujeres, o ambos). También se almacenarán las coordenadas actuales de posicionamiento del usuario (latitud y longitud). Esto servirá para calcular la distancia entre dos usuarios. Por último, el usuario podrá introducir, si lo desea, su fecha de nacimiento, una imagen (a modo de avatar), una lista con sus aficiones, y un texto con una breve descripción de sí mismo.

- Cada usuario tendrá una lista de **amigos**, que serán otros usuarios de la red social.

- Por otro lado, tenemos las **preguntas** que permitirán determinar la compatibilidad de dos perfiles distintos. La base de datos almacenará preguntas de variopinta temática (por ejemplo, *¿Te gustan los perros?*, *¿Cuál es tu color favorito?*, *¿Cuál es tu ideología política?*, etc.). Las preguntas son de tipo test, es decir, cada pregunta tiene una o varias posibles respuestas. El número de respuestas posibles puede variar de una pregunta a otra. No hay limitación en el número de respuestas posibles. Cuando un usuario decide contestar a una determinada pregunta, no sólo debe seleccionar una de las respuestas disponibles, sino que también indicará un número entre 0 y 10 indicando la relevancia que tiene dicha pregunta, según su opinión, a la hora de establecer la compatibilidad con otro usuario. Por ejemplo, un usuario puede decidir contestar a la pregunta *¿Cuál es tu color favorito?*, pero si además considera que la pregunta no le parece relevante, en tanto no le importaría que su pareja tuviese un color favorito distinto al suyo, asignará a esta pregunta un valor de 0. Por el contrario, la pregunta *¿Cuál es tu ideología política?* puede ser muy relevante para algunos usuarios de la red (en particular, para aquellos cuyo compromiso con la política es tal que les impediría tener una relación con otra persona que tuviese una ideología opuesta), mientras que a otros usuarios esta pregunta les puede parecer totalmente irrelevante.

- Por último, los usuarios de la red pueden comunicarse mediante **mensajes**. Existen tres tipos de mensajes:

  - Mensajes de texto propiamente dichos.
  - Solicitudes de amistad (al estilo de otras redes sociales, como *Facebook*).
  - Invitación a contestar a una determinada pregunta de las almacenadas en la BD.

  Independientemente del tipo de mensaje enviado se debe almacenar la fecha y hora de envío (un *timestamp*), y un valor booleano que indique si el mensaje ha sido leído o no por el destinatario. Además, obviamente, se guardará la información que requiera cada uno de estos tres tipos de mensajes.

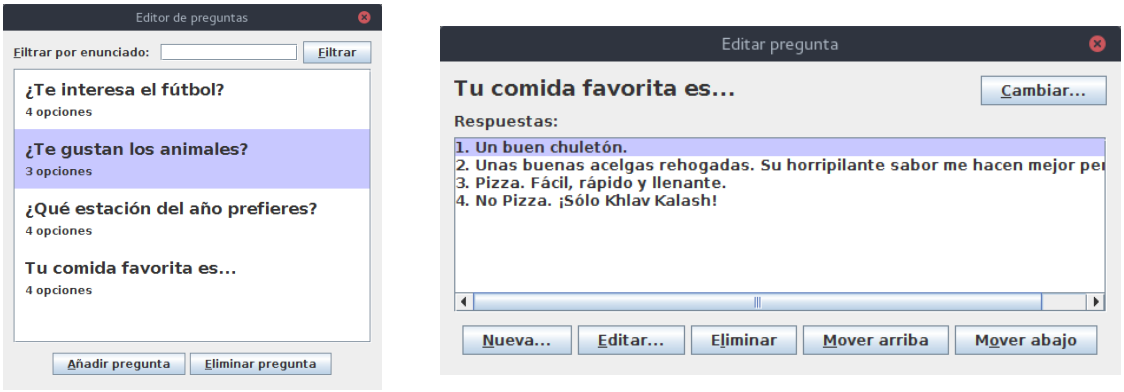


Figura 1: Interfaz de la herramienta de administración de preguntas.

Independientemente de lo especificado arriba, en aquellos casos en que no exista un identificador "natural" para un determinado tipo de entidad, se deberá añadir un campo numérico adicional a modo de identificador.

1 Diseño de la base de datos

En la primera entrega de la práctica nos centraremos exclusivamente en el primer objetivo de los enunciados arriba: el diseño de la base de datos. Para ello:

- Diseña el diagrama entidad-relación de la base de datos descrita anteriormente. Puedes hacerlo a mano y entregar una imagen escaneada, o utilizar herramientas como *Dia*, o *draw.io*.
- Crea una base de datos llamada `Practica_XXX`, donde `XXX` son los tres dígitos correspondientes a vuestro número de grupo. Puedes consultar este número en la lista de grupos de laboratorio del Campus Virtual de la asignatura. A continuación, crea las tablas necesarias para almacenar la información de la base de datos. Para ello aplica las técnicas (vistas en la asignatura de Bases de datos) de transformación de diagramas ER a modelos relacionales.

2 Diseño del modelo orientado a objetos y patrones de acceso a datos.

La segunda entrega tiene como objetivo aplicar la metodología de transformación de diagramas ER en modelos orientados a objetos, y utilizar los patrones de diseño de acceso a bases de datos vistos en clase. Esto último se pondrá en práctica mediante una herramienta de administración implementada en *Java* que permitirá a un usuario gestionar las preguntas contenidas en la base de datos (Figura 1). Para la realización de este apartado no tenéis que diseñar ningún componente de la interfaz gráfica, ya que se proporcionan en el Campus Virtual. Tan sólo tendréis que rellenar los métodos que realizan el acceso a la base de datos. Para ello:

- Descarga, a través del Campus Virtual, el proyecto plantilla `practica1Admin` correspondiente la herramienta de administración e impórtalo desde Eclipse. Se trata de un proyecto *Maven*.
- Crea las clases que conformarán el modelo de la aplicación que se creará en apartados posteriores. Estas clases deben materializar todas las entidades y relaciones contempladas en el modelo ER que has diseñado. Para ello ten en cuenta lo siguiente:

- Puedes modelar las relaciones entre clases mediante asociaciones unidireccionales o bidireccionales, según creas conveniente. En posteriores entregas podrás rectificar cualquier decisión que tomes ahora.
- Todas las clases han de situarse en el paquete `p1admin.model`.
- Los atributos de las clases han de ser privados, por lo que éstas últimas han de disponer de los métodos de acceso y modificación correspondientes.

  Indicación: Utiliza las opciones del menú *Source* de *Eclipse*.
- Ya se proporcionan las clases `Pregunta` y `Opcion` cuyas instancias representan, respectivamente, las preguntas y las posibles opciones de cada pregunta. No modifiques ninguna de estas clases. Estas clases ya vienen dadas debido a que se hace uso de ellas en la aplicación. Ambas clases suponen que la relación de pertenencia entre una pregunta y sus posibles respuestas (opciones) es de tipo uno-a-varios (es decir, una pregunta puede contener varias opciones), y que cada posible respuesta contiene un atributo numérico indicando su número de orden dentro de una pregunta. Las opciones se numeran desde 1 hasta N, donde N es el número de opciones de la pregunta.

- Si es necesario, modifica la base de datos para acomodar las decisiones de diseño sobre las preguntas y opciones mencionadas anteriormente. Es decir, que la relación entre ambas sea uno-a-varios y que las opciones tengan un número de orden.

- Dentro del paquete `adminDB` del proyecto crea una clase que implemente operaciones genéricas de *inserción*, *modificación* y *borrado* de filas en una tabla siguiendo el patrón *Data Accessor* visto en clase.

- Dentro de ese mismo paquete, implementa operaciones genéricas de *inserción*, *modificación* y *borrado* de filas en una tabla utilizando el patrón *Abstract Mapper* visto en clase. En la implementación de estas operaciones puedes (y deberías) llamar a los métodos del *Data Accessor* implementados anteriormente.

- De nuevo en el paquete `adminDB`, encontrarás una clase `DBFacade` que contiene funciones sin implementar. Esta clase no es más que una fachada[1] que será utilizada por el controlador de la aplicación para realizar las operaciones en la base de datos. Debes implementar las funciones de la fachada. La mayoría de ellas tan sólo han de realizar llamadas a los métodos genéricos de los *Mapper*. Por el contrario, otras operaciones de consulta son más específicas y requieren la realización de consultas *ad hoc*. En este último caso, implementa el código que realiza estas consultas en los *mappers* correspondientes, y haz la correspondiente llamada desde `DBFacade`. **No** hagas mención directa a nombres de tablas, columnas, etc. en la clase `DBFacade`.

  Indicación: Pon atención al problema de las $n + 1$ consultas. Una aplicación que haga un número de `SELECT`s proporcional al número de elementos recuperados será valorada negativamente.

- Ejecuta la herramienta de administración de preguntas mediante el método `main` de la clase `p1admin.Main`. Este método crea inicialmente una fachada "falsa" (clase `FauxDBFacade`), que es pasada al controlador principal de la aplicación:

[1] `https://en.wikipedia.org/wiki/Facade_pattern`

```
GenericDBFacade<Pregunta, Opcion> facade = new FauxDBFacade();
...
AllQuestionsController controller = new AllQuestionsController(model, facade);
```

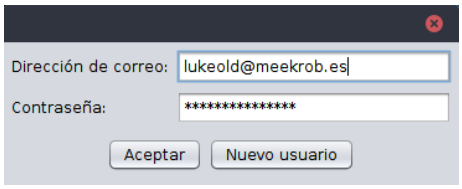
Esta implementación falsa no realiza ningún acceso a la base de datos. En su lugar guarda todas las preguntas en memoria utilizando dos `HashMap`. Para conectar la fachada que has realizado anteriormente con la aplicación deberás crear una instancia de ésta, y asignarla a la variable `facade` mostrada anteriormente. Al volver a ejecutar la aplicación, ésta llamara a los métodos de la fachada que has implementado, por lo que las preguntas y respuestas creadas se deberían guardar en la base de datos.

- Una vez que hayas comprobado que la herramienta funciona correctamente con tu fachada, crea un nuevo usuario de MySQL con nombre `AdminP1` y contraseña `PasswordP1`, y asígnale los permisos necesarios para que la herramienta pueda seguir funcionando cuando realiza sus conexiones utilizando este usuario. Puedes ejecutar directamente las sentencias DML (`CREATE USER`, `GRANT`, etc.) o asignar los permisos mediante la interfaz de *phpMyAdmin*. Las sentencias de asignación de permisos deberán guardarse en un script `.sql` distinto del que crea las tablas de la base de datos.

3 Preparando el terreno con *Hibernate*

Los restantes apartados de la práctica (incluyendo éste) tienen como objetivo el uso de *Hibernate* para acceder a la base de datos. **Todos los accesos a la BD que se implementen a partir de este punto deben hacerse mediante esta librería**, bien utilizando las funciones proporcionadas por `Session`, bien mediante HQL, según sea conveniente. Otro objetivo de la práctica es el desarrollo de interfaces gráficas de usuario para la visualización de datos. Por tanto, y al contrario que en la entrega anterior, no se proporcionará ninguna interfaz. En el Campus Virtual se darán indicaciones que os pueden ahorrar trabajo en este diseño.

- En primer lugar, descarga el proyecto plantilla `practica1User` desde el Campus Virtual, y renómbralo para que se llame `practica1UserXXX`, donde `XXX` es tu número de grupo. Copia las clases del modelo orientado a objetos que has definido en la entrega anterior a este proyecto.
- Modifica el fichero `hibernate.cfg.xml` poniendo la URL de tu base de datos dentro de la propiedad `connection.url`.
- Crea un usuario en MySQL de nombre `P1` y contraseña `PasswordP1`. Éste es el usuario que se utilizará en los siguientes apartados de la práctica para acceder a la BD. No olvides modificar los permisos de este usuario a medida que vayas completando los siguientes apartados de la práctica.
- Elimina las tablas de la BD que has creado en la entrega anterior. En el próximo paso vamos a intentar que Hibernate cree las tablas de la BD, de modo que el resultado sea igual a la BD que habéis definido en entregas anteriores. No gastéis tiempo y esfuerzo en intentar que los nombres de tablas y columnas coincidan exactamente con los de vuestra BD creada anteriormente. Lo importante es que la BD creada por *Hibernate* tenga la misma estructura que la vuestra.



(a) Ventana de acceso

(b) Ventana principal

Figura 2: Gestión básica de usuarios

- Añade las anotaciones de *Hibernate* necesarias en las clases del modelo orientado a objetos para que éste cree las tablas de la base de datos, de modo que el resultado tenga la misma estructura que la BD que habíais creado en la entrega anterior. Ejecuta el método `main` de la clase `CreateDB`. Este método creará un `SessionBuilder` con el fin de que se generen las tablas automáticamente.

  Importante: No olvides añadir las entradas `<mapping>` necesarias en el fichero `hibernate.cfg.xml`. Las clases entidad que no estén en este fichero no serán examinadas por *Hibernate*, aunque tengan el atributo `@Entity`.

4 Gestión básica de usuarios

La parte obligatoria de la aplicación consiste en implementar la funcionalidad de ingreso de usuarios al sistema, creación de nuevos usuarios, modificación de sus perfiles, y búsqueda de perfiles a partir de sus nombres.

- Al arrancar la aplicación se mostrará una ventana de acceso (*login*) como la de la Figura 2a. El usuario introducirá una dirección de correo y una contraseña. Si pulsa el botón `[Aceptar]` se comprobará que el nombre de usuario y contraseña introducidos aparecen en la BD. En caso afirmativo, se pasará a la ventana principal de la aplicación (Figura 2b). Si el usuario, tras introducir una dirección de correo y una contraseña, hubiese hecho clic en el botón `[Nuevo usuario]` de la ventana 2a, se abriría la ventana de edición de perfiles mostrada en la Figura 3.

- En la ventana de edición de perfiles de la Figura 3 el usuario podrá modificar todos sus datos excepto la dirección de correo. Al pulsar cada uno de los botones solicitará la información correspondiente. En la gran mayoría de los casos se puede utilizar un `JFileChooser` o un `JOptionPane` para esto. Los cambios introducidos por el usuario no se harán permanentes en la BD mientras no se pulse el botón de `[Guardar cambios]`.

- En la ventana principal (Figura 2b) aparecerá una lista de usuarios, que estará vacía inicialmente. Cuando el usuario escribe un texto en el `JTextField` situado al lado de la etiqueta `Filtrar por`

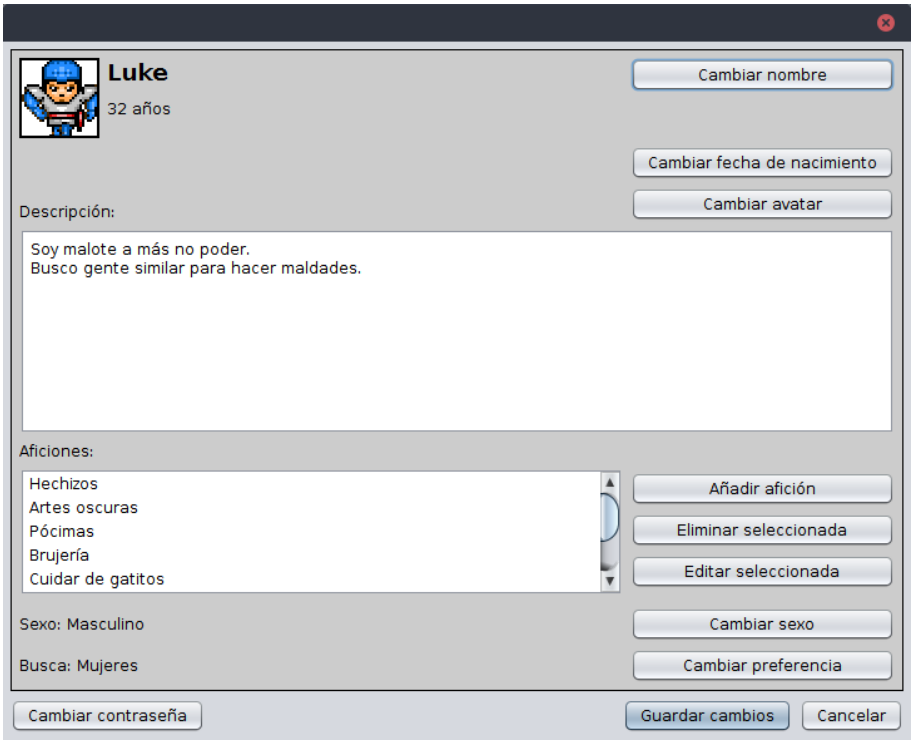


Figura 3: Modificación de perfil de usuario

`nombre`, se mostrarán en la lista aquellos perfiles cuyo nombre contenga dicho texto. Se descartarán de la lista aquellos usuarios cuyo género no se ajuste a los preferencias del usuario actual. De momento puedes ignorar los dos JCheckBox de la ventana y las pestañas `Preguntas` y `Mensajes no leídos`.

- Dentro de la ventana principal, cuando el usuario seleccione un elemento de la lista de usuarios y pulse en el botón `[Ver perfil seleccionado]`, se abrirá una ventana como la de la Figura 4. En ésta se mostrarán los datos del perfil que se ha seleccionado en la lista. La vista del perfil tiene el mismo aspecto de la Figura 3, pero sin los botones de edición. Por el momento puedes ignorar la etiqueta de texto con la distancia, el botón de solicitud de amistad, y el resto de pestañas de la ventana de la Figura 4, pues su funcionalidad se desarrollará en los siguientes apartados opcionales de la práctica.

5 Cálculo de la distancia

Comenzamos la primera extensión opcional de la práctica implementando las funciones que calculan la distancia entre dos usuarios. Para ello utilizaremos los datos de latitud y longitud de cada usuario.

¿Cómo calcular esta distancia? Todos sabemos que, la distancia entre dos puntos (x_1, y_1) y (x_2, y_2) en un plano viene dada por la fórmula $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$. Por desgracia, la tierra no es plana. Esto implica, por un lado, que no expresamos un determinado punto de la superficie terrestre mediante un par de coordenadas cartesianas, sino mediante un par (φ, θ) de coordenadas esféricas, donde $\varphi \in [-\pi/2, \pi/2]$ es la latitud y $\theta \in [-\pi, \pi)$ es la longitud. Por otro lado, la distancia entre dos puntos ya no

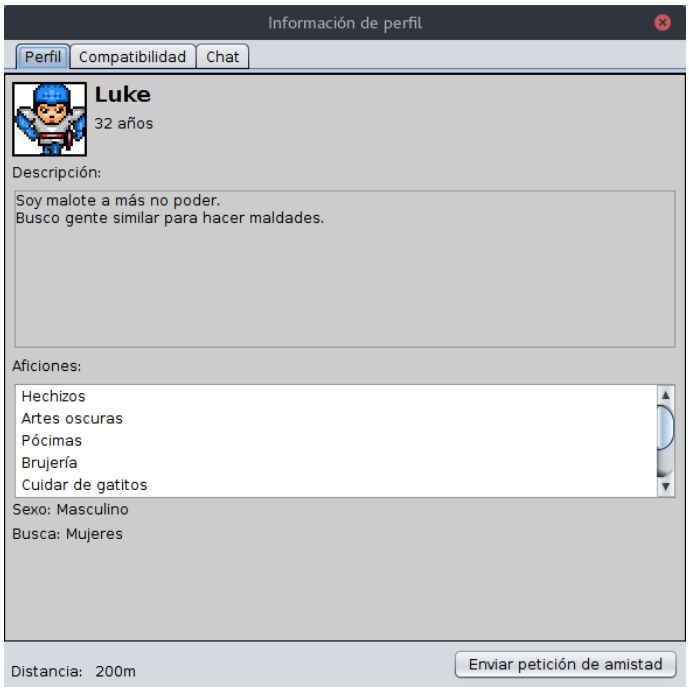


Figura 4: Visualización de perfil de usuario

es la longitud de la recta que los conecta, sino la longitud de la menor curva, situada en la superficie terrestre, que conecta ambos puntos. La *fórmula de Haversine* permite calcular la longitud de esta curva. Supongamos dos puntos con coordenadas (φ_1, θ_1) y (φ_2, θ_2), de modo que $\Delta\varphi = \varphi_1 - \varphi_2$ y $\Delta\theta = \theta_1 - \theta_2$. La distancia d entre estos dos puntos (medida sobre la superficie terrestre, en metros) se calcula mediante la siguiente fórmula:

$$d = R \cdot c \quad donde \begin{cases} R = 6371000 & \text{(radio aproximado de la Tierra en metros)} \\ c = 2 \cdot arc\ tan\left(\frac{\sqrt{a}}{\sqrt{1-a}}\right) \\ a = \sin^2\left(\frac{\Delta\varphi}{2}\right) + \cos\varphi_1 \cdot \cos\varphi_2 \cdot \sin^2\left(\frac{\Delta\theta}{2}\right) \end{cases}$$

Utiliza la función `atan2` de la clase `Math` para calcular el arco tangente de un cociente. Ten en cuenta que todas las coordenadas de longitud y latitud se introducen en radianes.

- Modifica la aplicación para que cada vez que un usuario ingrese en su cuenta de usuario se rellenen sus coordenadas de posicionamiento de manera aleatoria dentro de un determinado rango. Por ejemplo, puedes considerar la región que comprende la comunidad de Madrid y alrededores, tomando $\varphi \in [40°, 41'2°]$ y $\theta \in [3°, 4'5°]$.
- Modifica la vista de usuario de la Figura 4 (esquina inferior izquierda) para que se muestre la distancia entre el usuario actualmente ingresado en el sistema (el que está visitando el perfil) y el usuario del perfil mostrado en la vista. Para ello utiliza la fórmula de Haversine.
- Modifica la vista de la Figura 2b para que los usuarios se muestren ordenados en orden creciente de distancia. Si el JCheckBox de `Filtrar por nombre` está desactivado, se mostrarán en el

`JList` los veinte usuarios más cercanos de la base de datos. Si está activado, se mostrarán los veinte usuarios más cercanos de la base de datos cuyo nombre contenga la cadena contenida en el `JTextField` contiguo. En cualquier caso deberás utilizar la cláusula `ORDER BY` de HQL. Por simplicidad, puedes utilizar una aproximación similar a la distancia euclídea $(\Delta\varphi)^2 + (\Delta\theta)^2$ en lugar de la fórmula de Haversine para que la sentencia HQL sea más simple.

6 Contestar preguntas

En este apartado se ampliará la aplicación para que los usuarios puedan contestar a las preguntas de la base de datos.

- La pestaña `Preguntas` de la ventana principal (Figura 2b) contiene un componente `JList` y dos botones (Figura 5a). En esta lista deben mostrarse las veinte preguntas de la BD con mejor valoración media (en orden decreciente). Entendemos como 'valoración media' de una pregunta la suma de todas las valoraciones recibidas por todos los usuarios que han contestado a esa pregunta dividida entre el número de usuarios que han contestado a dicha pregunta. Utiliza la función de agregación `AVG` de HQL.
- Cuando el usuario hace clic en una de las preguntas del `JList` y pulsa el botón `[Responder]`, aparecerá una ventana como la de la Figura 5b. En ella el usuario puede contestar la pregunta seleccionando una de las opciones y valorar la relevancia de la pregunta. En el momento que pulse el botón `[Responder]` se guardará la contestación en la base de datos y se cerrará la ventana 5b, reactualizándose la lista de la Figura 5a. Si el usuario responde una pregunta que ya ha contestado previamente, se sobreescribirá su respuesta previa.
- Dentro de la ventana 5a, el botón `[Pregunta aleatoria]` debe mostrar una ventana como la de la Figura 5b, pero con una pregunta aleatoria, en lugar de la seleccionada en la lista. Puedes obtener una fila aleatoria de la tabla de preguntas mediante una consulta HQL que incluya la cláusula `ORDER BY RAND()`, y limitando el número de resultados a 1.

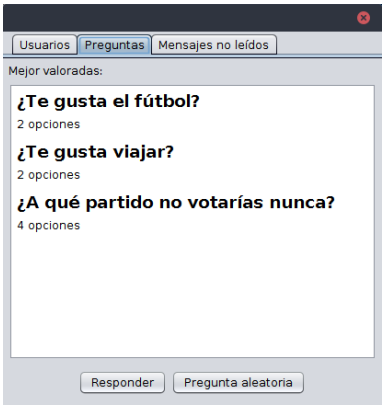
Por el momento ignora el botón `[Invitar a un amigo]` de la ventana 5b.

7 Cálculo de compatibilidad

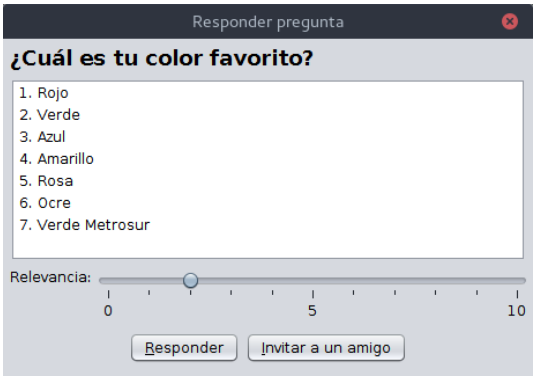
En esta extensión de la práctica se aborda el cálculo del nivel de compatibilidad entre dos usuarios de la red social. Este nivel de compatibilidad es un porcentaje que se calcula a partir de las preguntas que han sido contestadas por ambos usuarios. Veamos cómo se calcula:

Supongamos dos usuarios A y B. Cada usuario habrá respondido a un conjunto de preguntas de la base de datos. Algunas de ellas habrán sido respondidas por A y no por B, otras habrán sido respondidas por B y no por A, y otras habrán sido respondidas por ambos. Para el cálculo de la compatibilidad sólo nos fijaremos en este último conjunto. Así pues, tenemos un conjunto $\{x_1, \dots, x_n\}$ de preguntas que han sido respondidas por ambos usuarios. Denotando por $val_A(x_i)$ y $val_B(x_i)$ las respectivas valoraciones que los usuarios A y B han dado para la pregunta x_i, denotamos por M_{total} la suma de las valoraciones de ambos miembros en todas las preguntas que han respondido en común, no necesariamente con la misma respuesta.

$$M_{total} = \sum_{i \in \{1..n\}} (val_A(x_i) + val_B(x_i))$$



(a) Listado de preguntas



(b) Contestar una pregunta

Figura 5: Contestación de preguntas

Decimos que una pregunta x_i es un *acierto* si los usuarios A y B han coincidido en la respuesta a dicha pregunta. Los aciertos aumentan el nivel de compatibilidad entre A y B. Denotamos por $M_{acierto}$ la suma de las valoraciones de ambos miembros en todas las preguntas que han respondido en común y con la misma respuesta.

$$M_{acierto} = \sum_{\substack{i \in \{1..n\} \\ x_i \text{ acierto}}} (val_A(x_i) + val_B(x_i))$$

El porcentaje de compatibilidad viene dado por el resultado de la expresión $100 * (M_{acierto}/M_{total})$, redondeado al entero más cercano.

Realiza las siguientes modificaciones en la aplicación:

- Dentro de la ventana de la Figura 4 existe una pestaña llamada `Compatibilidad`. El contenido de esa pestaña se muestra en la Figura 6. Implementa una función que calcule el nivel de compatibilidad entre el usuario actual y el usuario que corresponde al perfil visitado, y lo muestre. Es posible que se necesite más de una consulta HQL para realizar este cálculo, pero presta atención al problema de las $n + 1$ consultas.
- En esta misma pestaña existe un `JList` que debe contener la lista de aficiones comunes entre el usuario actual y el usuario cuyo perfil está siendo visitado. No es necesario que calcules esta intersección mediante HQL. Puedes recuperar todas las aficiones de cada usuario mediante *Hibernate* y calcular la intersección entre las dos listas devueltas.

8 Interacción entre usuarios

Por último se implementará la funcionalidad relativa al aspecto social de nuestra red: relaciones de amistad, intercambio de mensajes e invitaciones a responder preguntas. Para ello:

- Implementa la funcionalidad del botón `[Enviar petición de amistad]` de la Figura 4. Por simplicidad supondremos que las peticiones de amistad se aceptan automáticamente (es decir, no requieren el consentimiento del receptor de la invitación). Cuando un usuario hace clic en este

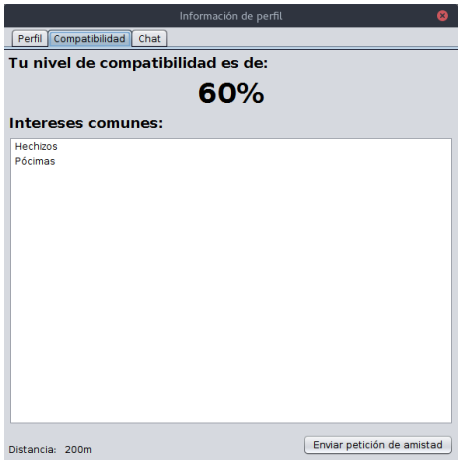


Figura 6: Compatibilidad entre usuarios

botón se añade la relación de amistad correspondiente, y se envía un mensaje al destinatario con la petición[2].

- Asocia un evento al JCheckBox de la Figura 2b titulado `Mostrar sólo amigos`, de modo que al seleccionarse sólo se muestren los amigos del usuario actual, en orden decreciente de distancia. Si la casilla `Filtrar por nombre` también está activada se mostrarán los amigos cuyo nombre contenga la cadena dada en el `JTextField`.

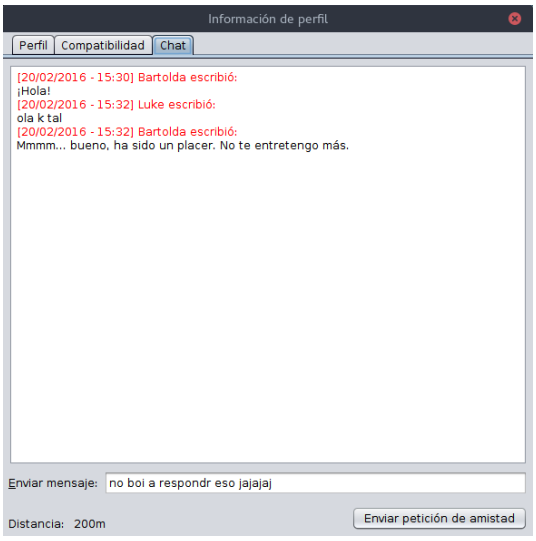
- Implementa la funcionalidad del botón `[Invitar a un amigo]` de la Figura 5b. Cuando el usuario haga clic sobre ese botón aparecerá un cuadro de diálogo como el de la Figura 7, cuyo funcionamiento es idéntico al panel de selección de usuarios de Figura 2b. Cuando el usuario pulse el botón `[Aceptar]` se guardará en la BD la invitación correspondiente.

- La pestaña `Chat` de la ventana de perfil de un usuario tiene el aspecto mostrado en la Figura 8a. En esta ventana se mostrarán los todos los mensajes de texto que se han enviado el usuario actual y el usuario del perfil visitado, en orden de fecha (desde el más antiguo al más reciente). Cuando el usuario introduzca un texto en el `JTextField` de la ventana y pulse *Enter* se enviará el mensaje de texto correspondiente.

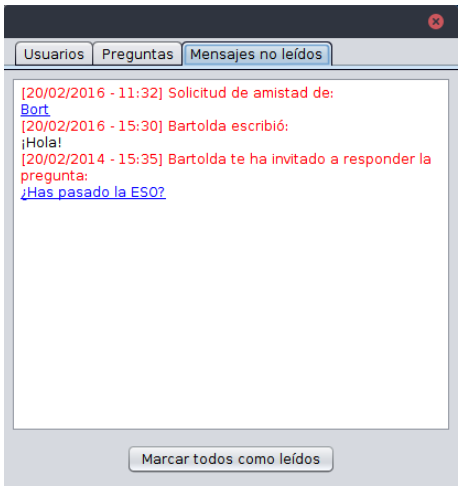
- Por último, implementa la visualización de la pestaña `Mensajes no leidos` de la ventana principal (Figura 8b). Aquí también aparece un `JTextArea` con todos los mensajes recibidos y no leídos por el usuario actual, independientemente de su tipo (texto, invitación a pregunta o solicitud de amistad) y del usuario del que provengan. En este `JTextArea` se mostrarán las peticiones de usuarios y las invitaciones a preguntas como hipervínculos. Cuando el usuario hace clic en un hipervínculo correspondiente a una petición de amistad, se mostrará la ventana de perfil correspondiente a dicho usuario (Figura 4). Cuando el usuario hace clic en un hipervínculo asociado a una invitación a pregunta, se abrirá una vista de pregunta (Figura 5b). El botón `[Marcar todos como leídos]` vaciará el `JTextArea` y modificará la BD marcando todos los mensajes del usuario como leídos.

[2] En este caso, es más una notificación de amistad que una petición propiamente dicha

Figura 7: Invitación a contestar una pregunta.



(a) Ventana de chat.



(b) Mensajes no leídos.

Figura 8: Envío y recepción de mensajes.

Instrucciones de entrega

Primera entrega: 29 de febrero de 2016.

Apartado 1. Entrega del diseño de la base de datos. Para ello debes subir en el Campus Virtual:

- Un fichero PDF con el diagrama entidad relación, bien sea una hoja escaneada, bien sea un diagrama creado con alguna herramienta.
- Una imagen con el esquema relacional. Para ello podéis realizar una captura de pantalla de la imagen generada por la herramienta Diseñador (*Designer*) de phpMyAdmin.
- El fichero `.sql` que genera las tablas de la base de datos. Este fichero no debe contener ninguna sentencia `CREATE DATABASE`.

Segunda entrega: 28 de marzo de 2016.

Apartado 2. Entrega del modelo orientado a objetos y las operaciones genéricas según los patrones *Data Accessor* y *(Abstract) Data Mapper*. Debes entregar en el Campus Virtual:

- Un fichero `.sql` que genera las tablas de la base de datos. De nuevo, no debe contener ninguna sentencia `CREATE DATABASE`.
- Un fichero `.sql` con las instrucciones necesarias para asignar los permisos al usuario `AdminP1` de MySQL.
- Un fichero comprimido (formato `.zip`) con el proyecto *Eclipse*.

Tercera entrega: 29 de abril de 2016.

Apartado 3. Entrega final, con las anotaciones Hibernate, la funcionalidad básica de gestión de usuarios, y (opcionalmente) las extensiones presentadas en los apartados 5 a 8. Uno de los componentes del grupo deberá subir:

- El fichero `.sql` que genera las tablas de la base de datos sin `CREATE DATABASE`.
- El fichero `.sql` con las instrucciones de asignación de permisos necesarias al usuario `P1` de MySQL.
- Un fichero comprimido (formato `.zip`) con el proyecto *Eclipse*. Si utilizas alguna librería externa que no has incluido como dependencia *Maven*, inclúyela en el directorio `lib` del proyecto.

A la hora de entregar la tarea aparecerá un cuadro de texto en el Campus Virtual donde deberás indicar:

- El nombre de los componentes del grupo que han realizado la práctica.
- En su caso, las extensiones opcionales de la práctica que habéis implementado.